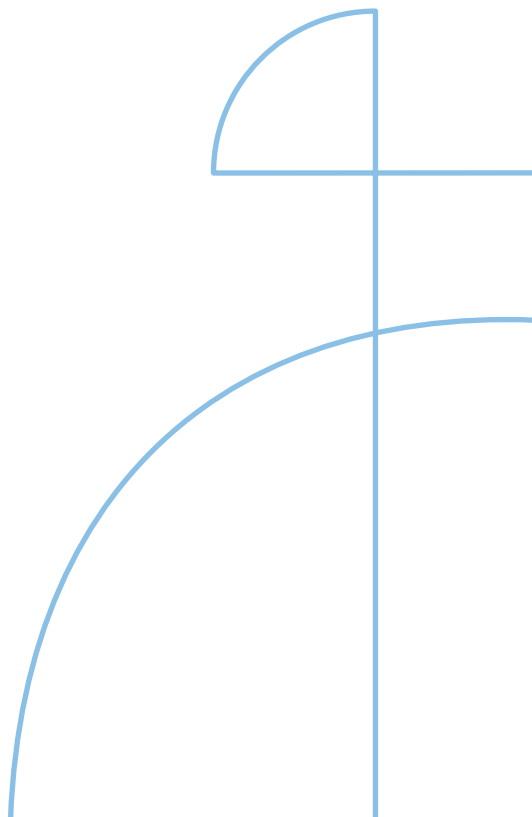
Doctoral Thesis in Computer Science

Matchings, Maxflows, Matroids

The Power of Augmenting Paths and Computational Models

JOAKIM BLIKSTAD

KTH ROYAL INSTITUTE OF TECHNOLOGY



Matchings, Maxflows, Matroids

The Power of Augmenting Paths and Computational Models

JOAKIM BLIKSTAD

Academic Dissertation which, with due permission of the KTH Royal Institute of Technology, is submitted for public defence for the Degree of Doctor of Philosophy on Wednesday the 27th of November 2024, at 14.00 in F3, Lindstedtsvägen 26, Stockholm.

Doctoral Thesis in Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden 2024

TRITA-EECS-AVL-2024:84
ISBN: 978-91-8106-091-1

Printed by: Universitetsservice US-AB, Sweden 2024

Abstract

Matchings, Maximum Flow, and Matroid Intersections are fundamental combinatorial optimization problems that have been studied extensively since the inception of computer science. A series of breakthroughs in graph algorithms and continuous optimization in the past decade has led to exciting almost-optimal algorithms for maximum flow and bipartite matching. However, we are still far from fully understanding these problems. First, it remains open how to solve these problems in modern models of computation, such as parallel, dynamic, online, and communication models. Second, as algorithms become more sophisticated in pursuit of efficiency, they often sacrifice simplicity, potentially obscuring valuable combinatorial insights. This raises a fundamental question: can we develop efficient algorithms that maintain the combinatorial nature of these problems, rather than relying on linear algebra and continuous methods?

This thesis returns to the classic augmenting paths framework—the original approach to matchings, maximum flow, and matroid intersection—with the goal of developing new efficient combinatorial algorithms. Our key contributions include the first combinatorial algorithm achieving almost-linear time for maximum flow on dense graphs, and the first subquadratic independence-query algorithm for matroid intersection. For modern computational models, our contributions include an improved online rounding scheme for fractional matching (leading to an optimal online edge coloring algorithm), a resolution of the query and communication complexity for bipartite matching, and the first sublinear-round parallel algorithms for matroid intersection.

Sammanfattning

Matchningar, Maximala Flöden och Matroidsnitt är grundläggande kombinatoriska optimeringsproblem som har studerats ingående sedan datorvetenskapens början. En serie genombrott inom grafalgoritmik och kontinuerlig optimering under det senaste årentiondet har lett till imponerande nästan optimala algoritmer för maximalt flöde och bipartit matchning. Vi är dock fortfarande långt ifrån att fullt förstå dessa problem. För det första återstår frågan om hur man kan lösa dessa problem i andra beräkningsmodeller, såsom parallell-, dynamisk-, online- och kommunikationsmodeller. För det andra, när algoritmer blir allt mer sofistikerade i deras effektivitetsträvan, offrar de ofta enkelhet, vilket potentiellt kan dölja värdefulla kombinatoriska insikter. Detta motiverar en grundläggande fråga: kan vi utveckla effektiva algoritmer som bevarar den kombinatoriska karaktären hos dessa problem, istället för att förlita sig på linjär algebra och kontinuerliga metoder?

Denna avhandling återgår till de klassiska augmenting-pathsalgoritmerna—det ursprungliga angreppssättet för matchningar, maximalt flöde och matroidsnitt—med målet att utveckla nya effektiva kombinatoriska algoritmer. Våra viktigaste bidrag inkluderar den första kombinatoriska algoritmen som uppnår nästan linjär tid för maximalt flöde på täta grafer, och den första subkvadratiska independence-query-algoritmen för matroidsnitt. För moderna beräkningsmodeller inkluderar våra bidrag förbättrade onlineavrundningsalgorithmer för fraktionell matchning (vilket leder till en optimal onlinealgoritm för kantfärgning), en lösning av query- och kommunikationskomplexiteten för bipartit matchning, och de första sublinjära parallella algoritmerna för matroidsnitt.

Acknowledgement

To my advisor...

Danupon Nanongkai.

Thank you for teaching me so much and guiding me so well during the past years. I truly could not imagine a better advisor.

To my wonderful co-authors...

Aaron Bernstein, André Nusser, David Wajc, Hanwen Zhang, Jan van den Brand, Mikkel Abrahamsen, Ola Svensson, Peter Kiss, Radu Vintan, Sagnik Mukhopadhyay, Ta-Wei Tu, Thatchaphol Saranurak, and *Yuval Efron*.

Working with you has been an absolute pleasure, and I look forward to the opportunity to do so again in the future.

Special thanks to...

Björn, for being a good friend since childhood.
Pratyush, for making Waterloo bearable.
Evangelos, for all your excitement.
Ioana, for not only reading my thesis, but making it better.
Tomasz, for the stunning hikes.
Yasamin, for all your energy.
Radu, for making EPFL extra special.
Ola and *Daniel*, for hosting me and making work so fun.
Sanjeev, *Eva*, *Rasmus*, and *Thore*, for your time and effort grading my thesis.

And a heartfelt thank you to all my friends around the world—you made these past few years incredibly fun and memorable. I am especially grateful to everyone at KTH, MPI-INF, BARC, EPFL, and CWI for the warm welcomes during my visits. I am truly proud to call you my friends.

And who could forget my family...

Ellen, *Ulf*, *Cissi*, *Birgitta*, and *Alexandra*.

Thank you for your inspiration and everlasting support.

Funding. My research has been supported by a Google PhD Fellowship; the Swedish Research Council under grand agreement Reg. No. 2019-05622; and the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No. 715672.

Publication List

At the time of writing, I have published ten papers in peer-reviewed conferences, and additionally have two manuscripts currently under review. In all of the following papers, I am a main contributor[1] throughout all phases, including conceptualization, formal analysis, and writing.

List of Works Included in the Thesis.

- **Paper A** [BBST24]
  Maximum Flow by Augmenting Paths in $n^{2+o(1)}$ Time.
  Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, Ta-Wei Tu.
  FOCS 2024 (invited to special issue)

- **Paper B** [BBMN21]
  Breaking the Quadratic Barrier for Matroid Intersection.
  Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, Danupon Nanongkai.
  STOC 2021

- **Paper C** [Bli21]
  Breaking $O(nr)$ for Matroid Intersection.
  Joakim Blikstad.
  ICALP 2021

- **Paper D** [Bli22]
  Sublinear-Round Parallel Matroid Intersection.
  Joakim Blikstad.
  ICALP 2022 (best student paper award)

- **Paper E** [BMNT23]
  Fast Algorithms via Dynamic-Oracle Matroids.
  Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, Ta-Wei Tu.
  STOC 2023

[1] Authors are sorted alphabetically, as is standard practice in theoretical computer science.

- **Paper F** [**BBEMN22**]
  Nearly Optimal Communication and Query Complexity of Bipartite Matching.
  Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, Danupon Nanongkai.
  FOCS 2022

- **Paper G** [**BK23**]
  Incremental $(1-\varepsilon)$*-Approximate Matching in* $O(\text{poly}(1/\varepsilon))$ *Update Time.*
  Joakim Blikstad, Peter Kiss.
  ESA 2023 (best student paper award)

- **Paper H** [**BSVW24b**]
  Simple and Asymptotically Optimal Online Bipartite Edge Coloring.
  Joakim Blikstad, Ola Svensson, Radu Vintan, David Wajc.
  SOSA 2024

- **Paper I** [**BSVW24a**]
  Online Edge Coloring is (Nearly) as Easy as Offline.
  Joakim Blikstad, Ola Svensson, Radu Vintan, David Wajc.
  STOC 2024 (invited to special issue)

List of Works Not Included in the Thesis.

- **Omitted Paper** [**ABNZ24**]
  Minimum Star Partitions of Simple Polygons in Polynomial Time.
  Mikkel Abrahamsen, Joakim Blikstad, André Nusser, Hanwen Zhang.
  STOC 2024

- **Omitted Paper** [**BSVW25**]
  Deterministic Online Bipartite Edge Coloring.
  Joakim Blikstad, Ola Svensson, Radu Vintan, David Wajc.
  SODA 2025

- **Omitted Paper** [**BT25**]
  Efficient Matroid Intersection via a Batch-Update Auction Algorithm.
  Joakim Blikstad, Ta-Wei Tu.
  SOSA 2025

The paper [ABNZ24] is omitted since it is a computational geometry paper, and hence out of topic for this thesis. Papers [BSVW25; BT25] are omitted since they are very recent works and were, at the time of writing, still under review.

Organization. Part I is a comprehensive summary of the main research questions, results, and technical contributions. Part I begins with an introduction (Chapter 1), and continues with the three topics: Maximum Flow (Chapter 2) covering Paper A; Matroid Intersection (Chapter 3) covering Papers B to E; and Matchings (Chapter 4) covering Papers F to I; ending with open questions (Chapter 5). The papers are attached in Part II.

Contents

Part I

Extended Synopsis

Chapter 1

Introduction

When we use computers to solve problems—whether it is finding the fastest route on a map, matching job seekers with employers, or organizing vast amounts of data—we must give the computers step-by-step instructions to tell them what to do. These instructions are called *algorithms*, and serve as the backbone of all our software. In the field of theoretical computer science, we study these algorithms at a deep level: developing new algorithms to solve problems efficiently with provable guarantees, showing that certain problems cannot be solved quickly (or at all), and trying to understand the fundamental limits and opportunities of what computers can achieve.

Despite decades of exciting advances in theoretical computer science, many fundamental computational problems remain far from fully understood. While efficient algorithms exist for some problems, others have persistently resisted our best efforts to find fast algorithms. Moreover, computing power is not the only resource constraint today: new challenges are, for example, how to effectively distribute computations across multiple devices and manage the communication channels between them. This motivates modern models of computation that captures these limitations and opportunities.

Overall, this points to a deeper question: why can certain problems be solved efficiently while others appear to require substantial computational resources? The ultimate quest for a unified theory—both across multiple problems and multiple models of computation—explaining these differences remains largely open. The challenge is further complicated by an emerging trend: as algorithms grow more sophisticated in their pursuit of efficiency, they often become increasingly complex.

1.1 Goals

With regard to the above mentioned challenges, the following ambitious question has driven the research behind this thesis.

Can we design efficient and simple algorithms in a unified way?

The goal of answering this question is threefold.

- Firstly, we want our algorithms to be **efficient**. In the classical sense, this means that they should run fast. In other models of computation considered in this thesis, efficiency is measured by low parallel depth or fast update time. The pursuit of efficiency is important because it directly impacts the usability, scalability, and energy costs of the algorithms. Efficient algorithms can handle larger datasets, perform under tighter time constraints, and utilize resources more effectively. These are all essential for real-world applications.

- Secondly, we want to design **unified** algorithms. We approached this in two key ways. In the **problem-unified** sense, algorithmic frameworks are developed to address a general problem, which can then be used to solve a wide range of related problems. This reduces the need to create distinct algorithms for every individual problem, promoting a more abstract, reusable approach. Similarly, **models-unified** algorithms work across different models of computation, such as classical, parallel, or communication models. These algorithms enable the same core ideas to be applied in various settings, thus bridging gaps between different models of computation and making solutions broadly applicable. Both perspectives aim for a high level of generalization and abstraction, and would lead to algorithms that are versatile across various problems and models. By developing unified algorithms, we can hope to better understand the essential characteristics that make certain problems solvable under certain resource constraints, and others not.

- Thirdly, we aim for **simplicity**. Out of all the possible techniques for solving the problems, we prioritize simple ones. Simple algorithms offer several advantages: they are easier to understand, which makes them more likely to be adapted and extended to new contexts. Moreover, simplicity of algorithms gives possibilities to teach to a wider audience, implement, and experiment to measure the algorithms' performance on real machines.

Designing efficient algorithms alone is a difficult task—for most problems in computer science, we do not know algorithms with provably optimal running times. Designing efficient algorithms that are also simple, and in a problem- or model-unified way, is hence even harder to achieve, and perhaps even impossible as such algorithms might not even exist. Nevertheless, this thesis strives to develop efficient, simple, and unified algorithms.

Matchings, Maxflows, Matroids. Towards the above goals, this dissertation focuses on three related fundamental combinatorial optimization problems: *Matchings*, *Maximum Flow*, and *Matroid Intersection*.

The study of these problems is originally motivated by their wide-ranging real-world application in fields such as resource allocation, route planning, and network optimization. In the field of theoretical computer science, they have

been central to the study of efficient algorithms since the inception of the field. Algorithmic improvements for these problems directly imply similar improvements to many other problems, making them well-suited for developing efficient problem-unified algorithms. These problems have also been studied from many different perspectives and serve as benchmarks for understanding the power of our current techniques in modern models of computations such as parallel, dynamic, online, and communication settings. While there have been many exciting advances in the study of matchings, maximum flow, and matroid intersection over the years, we remain far from fully understanding these fundamental problems. In the following, we narrow down our goals to three concrete research questions this thesis focuses on.

1. Combinatorial Algorithms vs. Continuous Optimization. The first algorithms for maximum flow, bipartite matching, and matroid intersection are all based on a combinatorial algorithmic framework called *augmenting paths*. For example, the famous Ford-Fulkerson [FF56], Edmonds-Karp [EK72], and Hopcroft-Karp [HK73] bipartite matching or maximum flow algorithms are all examples of augmenting path based techniques. While there have been many improvements in augmenting-path-based algorithms since their discovery, this progress has slowed down in recent decades. Instead, recent exciting research for these problems has focused on *continuous optimization methods*, culminating in the almost linear time algorithm for (minimum cost) maximum flow (or bipartite matching) by [CKLPGS22]. These continuous optimization algorithms are usually guided by an interior point method, and rely on dynamic data structures for graphs and linear algebra.

There are reasons to return and further study the augmenting paths framework. Firstly, for other problems such as matroid intersection or matching on general graphs, the state-of-the-art techniques are based on augmenting paths, and it is unclear if these problems allow for efficient continuous optimization-based methods. Secondly, augmenting-path algorithm are often simple to reason about since their progress is combinatorial, and improved augmenting path algorithms would possibly involve new combinatorial insights for the underlying problem. Thirdly, in practice, when run on real-world input, classic augmenting path algorithms often outperform[1] the continuous optimization methods, despite having worse theoretical guarantees.[2]

Thus studying augmenting path bases techniques poses a promising avenue to develop potentially simple, efficient, and unified algorithms. This discussion raises the question of whether continuous techniques are essential for solving maximum flow fast:[3]

[1] For example, Google's OR-tools [PF24] maximum flow implementation is based on the classic push-relabel augmenting paths approach [GT88]. Moreover, benchmarks show that versions of augmenting paths algorithms perform well in practice, see e.g. [GHKKTW15] and, to date, all the fast implementations for both the maximum flow and bipartite matching problems on the benchmarking website [Yos] are all based on augmenting paths.

[2] We do note, however, that it is not necessary that continuous methods are complicated, nor that augmenting-path algorithms are inherently simple or practical.

[3] With *combinatorial* we mean approaches, such as augmenting paths, that do not depend on continuous optimization or linear algebra techniques—we discuss this further in Chapter 2.

Question 1. *Can we design optimal maximum flow algorithms that are combinatorial?*

2. Another Problem-Unified Approach. While many problems reduce to maximum flow, other problems—including k-disjoint spanning trees [GW88; Gab91], colorful spanning tree [GS85], graphic matroid intersection [GS85; GX89], and many more (see Chapter 3)—have seen little to no progress in the past decades. It is unclear if these reduce to maximum flow[4] or not, and hence a natural question is:

Question 2. *Is there a problem-unified approach to simultaneously improve the bounds for problems such as k-disjoint spanning trees, colorful spanning tree, and graphic matroid intersection?*

These mentioned problems can be reduced to matroid problems. Hence we study the matroid intersection and union problems in order to develop problem-unified and efficient algorithms to answer the above question. Questions 1 and 2 are related to each other: studying the power of augmenting paths algorithms—both for maximum flow and matroid intersection—can shed light on both questions.

3. Models Of Computations. Modern computational models extend beyond the traditional paradigm (which focuses on running time) to address other real-world resource constraints and opportunities. For example, *dynamic* and *online* algorithms adapt to inputs changing over time. *Parallel* computing captures how well a problem can be split up and solved fast in distributed settings such as with multi-core processors. *Communication complexity* reflects the costs of data transfer, which is often the bottleneck in many distributed systems. Understanding how our fundamental problems—maximum flow, matchings, and matroid intersection—behave in other models, and how those models relate to each other, remains largely open. This motivates the following research question.

Question 3. *Under what resource limitations are our combinatorial optimization problems still efficiently solvable; specifically, can we develop matching algorithms that work in modern models of computation?*

Progress on the above three questions would further our understanding of the three fundamental problems we study. In the remainder of this chapter, we describe these problems and their connections briefly, and then summarize our main contributions towards these questions and our goals.

[4] After the original submission of this thesis, the paper [AK24] was announced, that, by using problem-specialized matroid intersection techniques, managed to reduce k-disjoint spanning trees to k maximum flow instances—hence achieving $O(\text{poly}(k)m^{1+o(1)})$ running time.

1.2 Matchings, Maxflows, Matroids

This section describes the three key topics of this thesis: *Matchings*, *Maximum Flow*, and *Matroid Intersection*. Figure 1.1 illustrates some examples of these problems. Formal definitions of the problems can be found in Chapter 2 (maximum flow), Chapter 3 (matroid intersection), and Chapter 4 (two matching problems: bipartite matching and edge coloring).

These problems are not only significant in their own right but are also interconnected with a wide array of other problems. As illustrated in Figure 1.2, numerous other combinatorial optimization problems, including bipartite matching, can be reduced to either maximum flow, matroid intersection problem, or both. Via these reductions, new algorithms for maximum flow and matroid intersection have the potential to simultaneously address multiple related problems. By studying these key problems, we hence aim to develop new problem-unified and efficient algorithms.

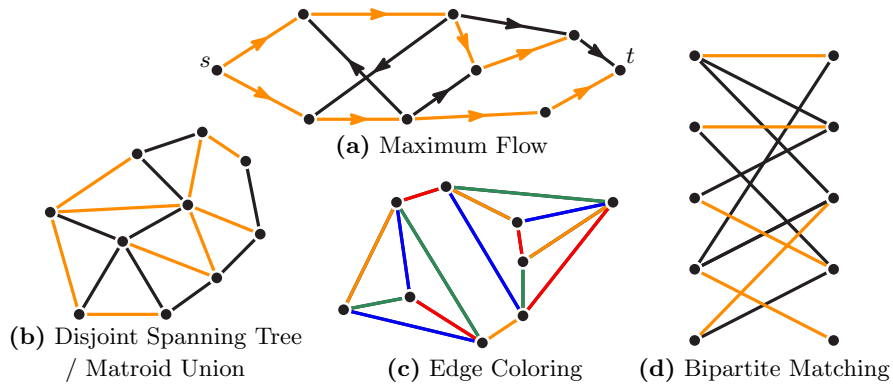


Figure 1.1: Illustrations of the graph problems we consider in this thesis. **(a)** A (unit-capacitated) *maximum flow* instance, with an optimal solution in orange. **(b)** A special case of the *matroid intersection/union* problem, namely decomposing a graph into two disjoint spanning trees (one in black, one in orange). **(c)** An *edge coloring* instance, the answer uses an optimal number of four colors. **(d)** A *bipartite matching* instance, with a maximum matching marked in orange.

Matchings. A matching in a graph is a set of edges where no two edges share the same vertex. The *bipartite matching* problem asks to find a maximum size matching in a bipartite graph, for example, matching job applicants to job positions. The bipartite matching problem plays a central role in this thesis together with two generalizations in different directions: maximum flow and matroid intersection. This thesis also studies another variant of the matching problem, namely *edge coloring*, which this thesis solves using improved matching algorithms. The edge coloring problem asks us to color the edges of a graph using a minimum number of colors such that no two edges sharing a vertex get the same color (or equivalently, covering a graph with the fewest number of matchings).

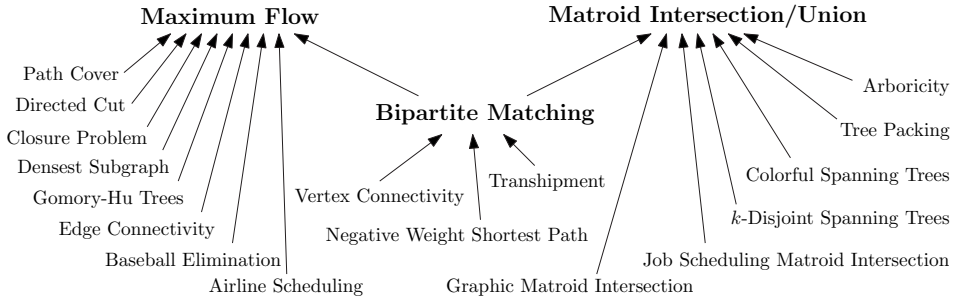


Figure 1.2: Many problems reduce to the maximum flow or matroid intersection problems, including the bipartite matching problem. A reduction (arrow) from problem A to problem B means that if one can (efficiently) solve problem B one can also (efficiently) solve problem A. Some of these reductions are only for the weighted variants (i.e., minimum cost maximum flow, max-cost bipartite b-matching, and weighted matroid intersections).

Maximum Flow. The *maximum flow* problem involves finding the maximum amount of flow that can be routed through a network from a source to a sink, respecting edge capacities. This fundamental problem has numerous applications in transportation, networking, and scheduling. For example, how much throughput of traffic that can be routed between two cities in a road network is an example of a maximum flow problem. Maximum flow serves as a cornerstone for many other graph algorithms and can be used to solve various related problems, including bipartite matching.

Matroid Intersections. Matroids are fundamental combinatorial objects that capture a notion of independence, for example spanning trees in graphs or linear independence in vector spaces. *Matroid intersection* (and the special case of *matroid union*) is a generalization of many combinatorial optimization problems, including bipartite matching, arborescences, and colorful spanning trees. It involves finding the largest set that is simultaneously independent in two different matroids defined on the same ground set. This problem captures many scenarios where one needs to optimize over two or more constraints at the same time. As a nontrivial example, it models the problem of finding two edge-disjoint spanning trees in a graph.

1.3 Summary of Contributions

This thesis designs novel and efficient augmenting-path algorithms for the maximum flow and matroid intersection problems. Further, we give algorithms for matchings—specifically for the bipartite matching and edge coloring problems—in various models of computations.

1.3.1 Augmenting Path Algorithms

This thesis tackles Question 1, i.e. the power of combinatorial algorithms, by developing augmenting-path-based algorithms for the maximum flow and matroid intersection problems. Our improved algorithms for matroid intersection together with a new *dynamic*-oracle model address Question 2 by showing a problem-unified approach to design efficient algorithms.

The basic idea behind augmenting paths is quite straightforward, see Figure 1.3 for an example of the concept applied to the maximum flow problem. How to apply augmenting paths to the bipartite matching and matroid intersection problems is similar.

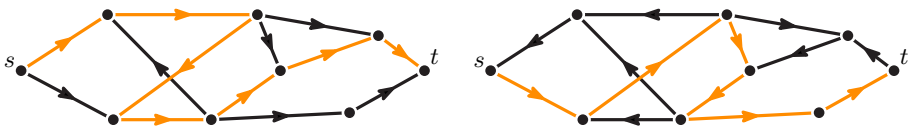


Figure 1.3: Execution of an augmenting-path algorithm for the maximum flow problem where all capacities are 1. On the left, one (s, t)-path is found, marked in orange. After augmenting along this path, the edges of this path become saturated so they are removed, and corresponding backward edges are added to form the residual graph, on the right. These backward edges allow for "un-sending" the flow along those edges, in case they were not part of the optimal solution. Then another augmenting path (marked in orange) is found in this residual graph. Note that this path uses previously reversed edges, effectively canceling out the old flow on those edges. The resulting flow of the two augmenting paths is exactly the flow depicted in Figure 1.1 (a).

The earliest augmenting path algorithms are often referred to as the Hungarian (or Kuhn-Munkres) algorithm [Kuh55; Mun57] that solves the assignment problem—also known as the min-cost perfect bipartite matching problem—or the Ford-Fulkerson algorithm [FF56] for the maximum flow problem. However, Jacobi might have invented a version of the Hungarian algorithm as early as 1836, although it was only published posthumously [JB65].[5] In the 1960's, Edmonds [EDVJ68; Edm70] showed a version of augmenting paths for the matroid intersection problem.

Next, we describe our key contributions new augmenting path algorithms for maximum flow and matroid intersection.

Combinatorial Maximum Flow. We make progress on Question 1 by developing a new *combinatorial* maximum flow algorithm. In particular, ours is the first such algorithm that runs in almost linear time on dense graphs, hence answering Question 1 when the input graph is dense. The new algorithm is based on iteratively finding augmenting paths and runs in $n^{2+o(1)} \log U$ time on directed n-vertex graphs with edge capacities in $\{1, 2, \ldots, U\}$. Even in unit capacitated graphs, this represents the

[5] Since this was before the invention of (theoretical) computer science as a field, there were no proofs of correctness or of polynomial running time.

first improvement in combinatorial maximum flow algorithms since the augmenting path based algorithms of [Kar73; ET75]. Similarly, for the special case of bipartite matching, our algorithm together with the independent work of [CK24a; CK24b], is the first combinatorial improvement since the well-known $O(m\sqrt{n})$-time Hopcroft-Karp algorithm [HK73].

In order to achieve this result, we develop a new variant of the classic push-relabel algorithm [GT88] that is guided by additional edge lengths as a hint, allowing it to focus on finding "short" augmenting paths. To find suitable edge lengths, we introduce and construct *directed expander hierarchies*, a generalization of a powerful graph decomposition technique previously for undirected graphs [Räc02; PT07; RST14; GRST21]. While our new maximum flow algorithm is conceptually simple, our construction of the expander hierarchy is complicated, although there is hope that future work can significantly simplify this step as directed expander decomposition techniques mature.

For further discussion about our maximum flow algorithm, and contributions towards Question 1, see Chapter 2 or Paper A [BBST24].

Faster Matroid Intersection and Union. Towards answering Question 2, this thesis gives several improved algorithms for the matroid intersection and the related matroid union problem. Since many other problems reduce to matroid intersection (recall Figure 1.2 on page 8), our improved algorithms give a problem-unified framework to solve many problems simultaneously, such as the k-disjoint spanning tree, colorful spanning tree, and graphic matroid intersection problems. All our matroid algorithms are based on a version of the augmenting paths framework, and are hence also combinatorial, in the spirit of Question 1.

Since it in general requires an exponential number of bits to represent a matroid, algorithms for matroid intersection usually assume query access to an oracle instead. As an conceptual contribution, we introduce a new *dynamic* oracle model for matroid optimization, giving a clearer understanding on how one can preserve efficiency of the algorithms through the aforementioned reductions from other problems.

We develop both new exact and approximation algorithms, in various oracle models. We also extend algorithms to work in the parallel setting. Our main contributions include

(i) the first matroid intersection algorithm using a subquadratic number of independence queries,

(ii) an improved rank-query algorithm for matroid union for dense matroids,

(iii) the first sublinear depth parallel matroid intersection algorithms, and

(iv) the first superlinear lower bounds.

Our results resolve open problems asked by, e.g., [Wel76; Har08b; LSW15; CLSSW19; CCK21].

As an example of a direct consequence of our algorithms, we obtain the first[6] algorithm that can find two disjoint spanning trees in a reasonably dense (n-vertex, m-edge) graph in near linear time, namely $\tilde{O}(m + n\sqrt{n})$ time.

Chapter 3 further discusses our new matroid intersection algorithms and Question 2. Details can be found in Paper B [BBMN21], Paper C [Bli21], Paper D [Bli22], and Paper E [BMNT23].

1.3.2 Models Of Computation

Towards answering Question 3, i.e. understanding the power and limitations of modern models of computations through the matching problem, this thesis studies our fundamental problems through the lens of online, dynamic, parallel, communication, and various query settings.

These computational models extend beyond the traditional sequential paradigm to address other resource constraints and opportunities. For example, *dynamic* and *online* algorithms adapt to changing inputs, as is often the case of real world data. *Parallel* computing captures how well a problem can be split up and solved fast in distributed settings such as with multi-core processors. *Communication complexity* reflect the costs of data transfer, which is often the bottleneck in many systems.

We already mentioned that our augmenting path algorithms for matroid intersection can also be implemented in the parallel setting as well in various query models, making for a model-unified algorithmic approach. Below, we mention additional contributions included in this thesis, specifically about the matching problems—edge coloring and bipartite matching—in different models of computation. More details and discussion about these contributions and Question 3 can be found in Chapter 4.

Communication and Query Complexity of Bipartite Matching. Communication complexity is the most basic model for capturing communication bottlenecks in computation. In the *communication setting*, the input (in our case the edges of a graph) is split up between two (or more) parties, and they are tasked with solving a problem together, using as little communication between them as possible. In *query models*, the underlying graph is unknown and one needs to ask queries to an oracle to figure out sufficient information about the graph to solve the given problem. Determining the communication and query complexities of bipartite matching are open problems that have been repeatedly raised in literature by, e.g., [HMT88; IKLSW12; DNO19; Nis21].

We show a surprisingly simple algorithm and tight lower bounds that resolve these open problems by (essentially) settling the communication and query complexity of the bipartite matching problem in at least six models (two-party communication, OR queries, XOR queries, AND queries, independent set queries, quantum edge queries) in a model-unified way. See Section 4.2 or Paper F [BBEMN22] for more details.

[6] [Qua23] independently and concurrently showed a similar result.

Incremental Approximate Matching. Dynamic algorithms extend the standard runtime analysis to the real-world settings where changes constantly occur. One challenging open problem is how fast one can recompute an approximate maximum matching undergoing changes to the underlying graph. In the *incremental dynamic* model, edges of a bipartite graph are revealed one by one. The goal is to maintain a $(1-\varepsilon)$-approximate (i.e., almost maximum) matching after each update. A trivial approach is to recompute the matching from scratch after each update, however this is wasteful of resources like computing power, time, and electricity. Instead a dynamic algorithm maintains some structure of the underlying graph to efficiently process each subsequent update to avoid recomputing from scratch.

We show a simple algorithm with constant update time $\text{poly}(1/\varepsilon)$ (i.e., completely independent on how large the graph is), for maintaining a $(1-\varepsilon)$-approximate matching. See Section 4.3 and Paper G [BK23] for more details.

Online Edge Coloring. Studying online algorithms is important for making decisions that cannot change. The *online* model is similar to the incremental dynamic model in that edges are revealed one by one. However, a crucial difference is that an online algorithm must make an irrevocable decision about the edge upon arrival. For example, in the online edge-coloring problem, the algorithm must assign a color to the arriving edge immediately and cannot change the assigned color later. In the incremental dynamic model, the goal was fast update time, but in the online model the goal is being able to produce a good solution (that is, one that is competitive with what an offline algorithm with unlimited resources can produce). This is usually challenging, since the algorithm must make these irrevocable decisions before seeing the full input.

We resolve a longstanding conjecture of [BMN92] by showing that there is an online algorithm that uses at most $(1+o(1))\cdot\Delta$ colors (i.e., almost the same number of colors as an offline algorithm) for graphs of maximum degree $\Delta \geq \omega(\log n)$. Our edge-coloring algorithms, discussed in Section 4.4 (and in Paper H [BSVW24b] and Paper I [BSVW24a]), and their analyses, are surprisingly simple, and extend to other related problems such as list edge coloring or rounding spread out fractional matchings online.

Organization

The following three chapters each focuses on one of the three topics: maximum flow, matroid intersection, and matchings. Specifically, Chapter 2 presents a new augmenting-path-based combinatorial algorithm for maximum flow. Chapter 3 presents a sequence of advances of augmenting path based algorithms for matroid intersection. Chapter 4 presents our result on matchings in different models of computation, including communication and query complexity (Section 4.2), dynamic (Section 4.3), and online (Section 4.4). To conclude, we end with discussions of open questions and future directions in Chapter 5.

Chapter 2

Maximum Flow

In this chapter we outlines our combinatorial $n^{2+o(1)}$ algorithm for the maximum flow problem from Paper A [BBST24]. We begin by defining the problem.

Maximum Flow.
Given a directed n-vertex, m-edge graph $G = (V, E)$ with positive integer edge capacities $\boldsymbol{c} : E \mapsto \{1, 2, \ldots, U\}$ and two special vertices $s, t \in V$, a *flow* $\boldsymbol{f} : E \mapsto \mathbb{R}$ satisfies:

- capacity constraints $0 \leq \boldsymbol{f}(e) \leq \boldsymbol{c}(e)$, and
- conservation of flow at each vertex $v \in V \setminus \{s, t\}$, that is the outgoing flow equals the incoming flow, i.e., $\sum_{(v,u)\in E} \boldsymbol{f}(v, u) = \sum_{(u,v)\in E} \boldsymbol{f}(u, v)$ for all $v \in V \setminus \{s, t\}$.

The *maximum flow* $\boldsymbol{f}$ is the one that routes the largest flow from the source s to the sink t, i.e., the flow maximizing $\sum_{(s,u)\in E} \boldsymbol{f}(s, u) - \sum_{(u,s)\in E} \boldsymbol{f}(u, s)$.

Efficient algorithms for calculating maximum flows have been a key focus in algorithmic research, inspiring various approaches including graph sparsification, dynamic data structures, and continuous optimization techniques. Maximum flow algorithms have broad applications, including in areas such as bipartite matching, vertex connectivity, directed cut, and the construction of Gomory-Hu trees (see e.g., [GH61; LP20; LNPSY21; CLNPSQ21; CHLP23; ALPS23]). Consequently, studying efficient algorithms for the maximum flow problem provides a solid foundation for developing a unified approach to solving multiple problems with the same framework.

Recent years have witnessed exciting advancements in maximum flow algorithms based on continuous optimization and dynamic graph data structures (see e.g., [ST04; CKMST11; KMP12; She13; KLOS14; Pen16; She17; ST18; DS08; Mad13; LS14; Mad16; LS20; KLS20; LS14; KLS20; DGGP19; BBPNSSS22; CGHPS20; BLNPSSSW20; BLLSSSW21; GLP21; BGJLLPS22]), culminating with the break-

through work of [CKLPGS22]. They show the first maximum flow[1] algorithm that runs in almost linear time, $m^{1+o(1)}$.

Combinatorial Approaches. While recent advancements have achieved (almost) optimal time complexity for maximum flow algorithms, these solutions are complex both conceptually and technically. Modern continuous optimization methods modify flow solutions without clear combinatorial interpretations, and they rely on intricate dynamic graph data structures.

In contrast, *combinatorial* algorithms do not rely on sophisticated linear algebra or continuous methods. Instead, they work with discrete structures of the underlying (graph) problem.

Earlier flow algorithms, while less efficient, employed the more intuitive and combinatorial *augmenting paths* framework. For computing maximum flows, this framework was initially proposed by Ford and Fulkerson [FF56]. In this approach, algorithms iteratively identify an augmenting path (or a set of such paths) within the residual graph, then increase the flow along this path by an amount equal to the path's bottleneck capacity (recall Figure 1.3). This straightforward concept sparked the development of several algorithms over the subsequent decades. These include the shortest augmenting paths method [EK72], the concept of blocking flows [Kar73; Din70; GR98; HK73], the push-relabel algorithm [GT88; Gol08], and graph sparsification techniques [KL15].

However, progress on augmenting path (or combinatorial) approaches to the maximum flow problem have slowed down in the recent decades in favor of the continuous optimization approaches (see Table 2.1 for a summary of maximum flow algorithm development). It is however unclear if these continuous techniques are indeed necessary to develop fast maximum flow algorithms. This raises Question 1, which we repeat below.

> **Question 1.** *Can we design optimal maximum flow algorithms that are combinatorial?*

Recent research has explored this direction, aiming to develop improved *combinatorial* algorithms for various other problems, such as boolean matrix multiplication [AFKLM24; BW09] and bipartite matching [CK24a; CK24b].

For the maximum flow problem, we tackle Question 1 by revisiting the augmenting paths framework. Indeed, augmenting path algorithms are commonly implemented in practice and often perform well on real-world inputs, despite having worse theoretical guarantees than the more modern continuous optimization techniques. Another reason for studying on augmenting path algorithms is that it is uncertain whether efficient continuous optimization methods exist for other problems, such as matroid intersection and matching on general (non-bipartite) graphs, where the best current running times are achieved using variations of the augmenting path-based

[1] Their algorithm also works for the more general minimum cost maximum flow problem.

approach. Improved augmening path algorithms for the maximum flow problem hence have a chance of being adapted to give faster algorithms for general matching or matroid intersection.

2.1 Contribution

In Paper A [BBST24], we make considerable progress towards Question 1 by designing an augmenting-path-based algorithm whose running time is almost optimal for dense graphs.

Theorem 2.1.1. *There is an augmenting path-based randomized algorithm that, given a directed graph with n vertices and edge capacities from $\{1, 2, \ldots, U\}$, with high probability*[2]*, computes a maximum flow in $n^{2+o(1)} \log U$ time.*

Our algorithm represents the first improvement in combinatorial approaches since the $\tilde{O}(m \cdot \min\{\sqrt{m}, n^{2/3}\})$ algorithms developed in the 1970s by [ET75] and [Kar73] for unit-capacitated graphs, and in the 1990s by [GR98] for general capacities. See also Table 2.1 for a comparison with other combinatorial and non-combinatorial algorithms.

Year	Comb?	Authors	Running Time	Comments
1955	✓	[FF56]	$O(m \cdot \|\text{answer}\|)$	
1970	✓	[EK72]	$O(m^2 n)$	
1970	✓	[Din70]	$O(mn^2)$	"Blocking Flow"
1975	✓	[ET75]	$O(mn^{2/3})$	(unit-capacity)
1988	✓	[GT88]	$\tilde{O}(mn)$	"Push-Relabel"
1998	✓	[GR98]	$\tilde{O}(m \cdot \min\{\sqrt{m}, n^{2/3}\})$	
2014		[LS14]	$\tilde{O}(m\sqrt{n})$	
2020		[KLS20]	$m^{4/3+o(1)}$	(unit-capacity)
2021		[BLLSSSW21]	$\tilde{O}(m + n\sqrt{V})$	
2021		[GLP21]	$\tilde{O}(m^{3/2-1/328})$	
2022		[CKLPGS22]	$m^{1+o(1)}$	
2024	✓	**Ours**	$n^{2+o(1)}$	Paper A [BBST24]

Table 2.1: A non-exhaustive history of maximum flow algorithms. Running times are stated for n-vertex, m-edge graphs with polynomially (in n) large capacities. Combinatorial algorithms—in this case, all based on the augmenting paths framework—are marked by a check mark ✓. The other algorithms are based on continuous optimization techniques.

In an independent line of similar work, [CK24a; CK24b] presented a combinatorial $n^{2+o(1)}$ algorithms for computing the maximum bipartite matching (and

[2] *With high probability* means with probability $1 - 1/n^c$ for an arbitrarily large constant c.

consequently, also unit-vertex-capacitated maximum flow), a special case of the maximum flow problem. Their techniques are based on dynamic shortest-path data structures and multiplicative weights update and, hence, different from ours. For bipartite matching, [CK24a; CK24b] and our algorithm are the first improvement, in sufficiently dense graphs, in combinatorial algorithms since the classic $O(m\sqrt{n})$ time Hopcroft-Karp [HK73] algorithm. The $O(m\sqrt{n})$ time bound of the Hopcrof-Karp algorithm comes from a natural balancing out two terms: finding all "short" augmenting paths up to length ℓ (setting $\ell = \sqrt{n}$) in $O(m\ell)$ time, and then the remaining n/ℓ many "long" augmenting paths one by one, each in $O(m)$ time, for a total of $O(m\ell + mn/\ell)$ time. To enable our new augmenting paths algorithm to achieve near-linear time on dense graphs, we move away from this balancing of short and long paths. Instead, we introduce a new notion of "lengths" that depends on the structure of the input graph, and use these lengths to guides our augmenting paths.

Sketch of Techniques. Our algorithm strictly follows the augmenting path framework in that it always sends *integral* flows along paths. Note that it suffices to design a constant- or even $1/n^{o(1)}$-approximate flow algorithm for directed graphs, as the exact algorithm then follows by repeating the approximate algorithm $n^{o(1)}$ times on the residual graph.[3] See also the technical overview Section A.2 in Paper A [BBST24] for a more comprehensive overview of our approach.

Our first technical contribution is a new version of the classic push-relabel algorithm [GT88] that also supports additional edge weights as inputs. These edge weights serve as hints on how to list augmenting paths efficiently, allowing the algorithm to focus on finding *short* paths (with respect to these edge weights); edges with higher weight are relabeled less often.

These additional edge weights $\boldsymbol{w} : E \to \mathbb{Z}_{\geq 1}$ need to satisfy certain properties for them to be useful. We call the edge-weight function $\boldsymbol{w}$ *good* if the following are satisfied:

- **Running-Time-Property:** $\sum_{e \in E} \frac{n}{\boldsymbol{w}(e)} \leq n^{2+o(1)}$. This is required to guarantee the running time of the push-relabel algorithm.
- **Approximation-Property:** There must exist a $1/n^{o(1)}$-approximate flow $\boldsymbol{f}^\star$ which is "short" with respect to the edge weights, i.e., the average length of unit-flow paths in $\boldsymbol{f}^\star$ is small: $\sum_{e \in E} \boldsymbol{w}(e)\boldsymbol{f}^\star(e) \leq n^{1+o(1)}$. This is required to guarantee that the flow $\boldsymbol{f}$ computed by our push-relabel algorithm is at least a constant fraction of $\boldsymbol{f}^\star$.

Finding *good* edge-weights on directed acyclic graphs (DAG) turn out to be easy: set $\boldsymbol{w}(u, v) = |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$ where $\boldsymbol{\tau}$ is a topological order. This, together with our

[3] This is in contrast to undirected graphs, where, although efficient approximations are known [She13; KLOS14; Pen16; She17; ST18], the residual graph of becomes directed, not allowing the approximate flow algorithm to be bootstrapped to an exact one.

new weighted push-relabel, immediately gives a simple, constant-approximate flow algorithm in $\tilde{O}(n^2)$ time on DAGs. See Section 2.1.1 for further discussion on our weighted push-relabel algorithm, and why the above $\boldsymbol{w}$ is *good*.

Directed Expander Hierarchy. What remains is to find *good* edge weights on general directed graphs. It turns out that setting $\boldsymbol{w}(u, v) = |\boldsymbol{\tau}'(u) - \boldsymbol{\tau}'(v)|$ where, informally, $\boldsymbol{\tau}'$ is something like a "pseudo-topological-order" works. To find this ordering of vertices $\boldsymbol{\tau}'$, we introduce and construct *directed expander hierarchies*.

Expander decomposition is a powerful technique for flow, cut, and shortest path problems. In the directed setting (see e.g., [BPS20; HKPW23; SP24]), it identifies a small set of *back-edges* B such that each strongly connected component of $G \setminus B$ is an *expander*. An *expander* is a well-connected graph of low diameter (see Paper A [BBST24] for a full definition), that allow for easy routing of flows. As an example, random graphs are expanders.

In an expander hierarchy, there are multiple ($\log n$ or fewer) levels of expander decomposition (to recursively handle the back edges B). While there are successful variants of expander hierarchies in undirected graphs [Räc02; PT07; RST14; GRST21], we believe that ours is the first application of them in directed graphs. Given a directed expander hierarchy of a graph, we show a simple algorithm which finds a suitable "pseudo-topological-order". To demonstrate that this induces a *good* edge weights, we establish a new trade-off between path length and congestion when rerouting flow on expanders.

Additionally, to prove the existence of a sparse level cut—which is necessary when constructing the expander hierarchy—we introduce a novel approach to the *directed expander pruning* problem, as explored in [BPS20; HKPW23; SP24]. The traditional version of pruning certifies that, after a few edges in a expander X are erased, there is a small (proportional to the number of removed edges) *pruned* part P such that $X \setminus P$ still remains an expander. We extend this technique to handle *path-reversal* updates—an operation that naturally arises in residual graphs during the reversal of augmenting paths. Interestingly, reversing an entire path has roughly the same effect on pruning as erasing a single edge. This insight allows us to prove that the directed expander hierarchy remains robust under flow augmentation.

Constructing our directed expander hierarchy can be done in a top-down manner as in the undirected case [RST14], *however, this uses a maximum flow subroutine, which we cannot assume*. Instead, we show a bottom up construction where, to build the ℓ'th level of the hierarchy, we bootstrap our weighted push-relabel algorithm and feed it edge weights from the lower levels. This comes with several technical challenges, and unfortunately introduces a large amount of complexity. We hope that this can be simplified as directed expander decomposition techniques matures.

Summary. Unlike recent advancements, our combinatorial maximum flow approach does not depend on continuous optimization or complex dynamic data structures. Moreover, it opens the door to a highly implementable $\widetilde{O}(n^2)$-time

deterministic algorithm, contingent on future simplifications in the construction of directed expander hierarchies. Paper A [BBST24] is largely self-contained, with the only external components being standard graph algorithms (e.g., topological sorting, Dijkstra's algorithm, and link-cut trees [ST83]), and Louis's cut-matching game [Lou10] used in our expander decomposition. We also hope that some of the new tools we developed—including the *weighted push-relabel algorithm*, *directed expander hierarchy*, and *expander pruning under path-reversals*—will have broader applications in the future.

2.1.1 Weighted Push Relabel

Let us first recall a simple version of the classic push-relabel [GT88] algorithm called *augment-relabel* (see e.g., [Gol08]). This version only pushes whole augmenting paths, and hence always keeps track of a flow (as opposed to a preflow). The algorithm begins by computing $\boldsymbol{\ell}(v)$ as the distance from vertex v to the sink t. Call an edge (u, v) *admissible* if $\boldsymbol{\ell}(u) = \boldsymbol{\ell}(v) + 1$. Now, the algorithm will trace a shortest augmenting path by arbitrarily following admissible edges starting from the source s until they reach the sink t. The algorithm continues by recomputing the distance labels $\boldsymbol{\ell}(v)$ in the residual graph, and then augments along another path, and so on.

A crucial observation is that after a path-reversal of a shortest augmenting path, the distance from any vertex v to the sink can only increase. That is, the labels $\boldsymbol{\ell}(v)$ are monotonically increasing. The augment-relabel algorithm thus uses the following simple *relabeling* strategy to recompute the labels: while a vertex $u \neq t$ with no admissible outgoing edges exists, increment $\boldsymbol{\ell}(v)$ by one. When $\boldsymbol{\ell}(v) > n$ we can ignore v since we know it cannot reach the sink anymore.

In total there are at most n^2 relabel operations, since each vertex will visit each distance level at most once. The running time is $\tilde{O}(nm)$ if implemented carefully, as each edge will be tried once as an admissible edge on each level, and, in the capacitated setting, flow can be pushed efficiently over dynamic trees [ST83].

Weighted Push-Relabel. Our new algorithm follows a similar approach to the above one. The key difference is that the distances are computed with respect to our additional edge weights $\boldsymbol{w} : E \to \mathbb{Z}_{\geq 1}$. In particular, we call an edge $e = (u, v)$ *admissible* if $\boldsymbol{\ell}(u) \geq \boldsymbol{\ell}(v) + \boldsymbol{w}(e)$. Again, there are at most $O(n^2)$ relabel operations, but we will beat the $O(nm)$ time bound for handling the edges. Indeed, for an edge $e = (u, v)$, we only consider it $\approx \frac{n}{\boldsymbol{w}(e)}$ times in the algorithm—when either of u or v enters a level which is a multiple of $\frac{\boldsymbol{w}(e)}{10}$. This means that the algorithm allows some slack in the edge lengths and only maintains approximate distances as $\boldsymbol{\ell}$.

The implementation turns out to be similar to the standard push-relabel or augment-relabel algorithm. The algorithm runs in $\tilde{O}(n^2 + \sum_{e \in E} \frac{n}{\boldsymbol{w}(e)})$ time, which is small assuming the *running-time-property*. To prove the approximation factor, assuming the *approximation-property*, is also easy: if the algorithm sends less than,

say, a $\frac{1}{10}$-fraction of the flow, then, by an averaging argument, there must remain a reasonably short flow path.

Good Edge Weights for DAGs. Let $\boldsymbol{w}(u,v) = |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$, where $\boldsymbol{\tau}$ is a topological order of the input directed acyclic graph $G = (V, E)$. That is, $\boldsymbol{\tau}$ is a bijection from V to $\{1, 2, \ldots, n\}$, so that $\boldsymbol{\tau}(u) > \boldsymbol{\tau}(v)$ for all edges $(u, v) \in E$. We argue that this is a *good* weight function. Indeed, the *running-time-property* holds since

$$\sum_{e \in E} \frac{n}{\boldsymbol{w}(e)} \leq \sum_{u \in V} \sum_{(u,v) \in E} \frac{n}{|\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|} = O(n^2 \log n),$$

as the inner sum is at worst a harmonic sum. The *approximation-property* is also easy: all paths in a DAG has length at most n according to $\boldsymbol{w}$, since traversing some edge e brings us forward by $\boldsymbol{w}(e)$ in the topological order. Since all paths have length at most n, so do all paths in the maximum flow. Combined with the new weighted push-relabel algorithm, this gives a simple constant-approximate flow algorithm for DAGs running in $\tilde{O}(n^2)$ time.[4]

Good Edge Lengths in General Graphs. We note that if we already had a maximum flow $\boldsymbol{f}^\star$ of the graph, we can assume that the support of $\boldsymbol{f}^\star$ induces a DAG (any cycles can be cancelled in the flow). The topological order of this DAG would then give a good edge lengths for the full graph. However, we can of course not assume that we already know a maximum flow in order to solve the maximum flow problem. Instead we introduce a notion of directed expander hierarchies (and show how to bootstrap our weighted push-relabel to compute them) that will give us a good enough approximation of the topological order of an (unknown) maximum flow, hence giving us good edge lengths. See Paper A [BBST24] for more details.

[4] However, the residual graph might no longer be a DAG, so to get exact flow on DAGs one still need to find *good* edge lengths for general graphs.

Chapter 3

Matroid Intersection and Union

This chapter outlines recent advances in matroid intersection algorithms (Paper B [BBMN21], Paper C [Bli21], Paper D [Bli22], Paper E [BMNT23]).

As mentioned in the previous chapter, via reductions to (minimum cost) maximum flow, exciting progress has been made for many graph problems such as maximum matching, vertex connectivity, directed cut, and Gomory-Hu trees. However, some other problems—such as k-disjoint spanning trees [GW88; Gab91], colorful spanning tree [GS85], arboricity [Gab95], spanning tree packing [GW88], graphic matroid intersection [GS85; GX89], and simple job scheduling matroid intersection [XG94]—have seen little to no progress in the past decades. It is unclear if these problems allow for efficient reductions to (minimum cost) maximum flow. This motivates Question 2, which we restate below.

> **Question 2.** *Is there a problem-unified approach to simultaneously improve the bounds for problems such as k-disjoint spanning trees, colorful spanning tree, and graphic matroid intersection?*

Many of these problems can be modeled as **matroid problems**, making it a natural avenue of study to improve bounds for these problems all at once in a unified manner. We begin by formally defining matroids and the optimization problems we consider in this chapter.

Matroid. A *matroid* $\mathcal{M} = (U, \mathcal{I})$ is defined by a tuple consisting of a finite *ground set* U and family of *independent sets* $\mathcal{I} \subseteq 2^U$ such that the following properties hold.

- **Non trivial:** The empty set is independent, that is $\emptyset \in \mathcal{I}$.
- **Downward closure:** If $S \in \mathcal{I}$, then any subset $S' \subseteq S$ is also in $\mathcal{I}$.
- **Exchange property:** For any two sets $S_1, S_2 \in \mathcal{I}$ with $|S_1| < |S_2|$, there exists an $x \in S_2 \setminus S_1$ such that $S_1 + x \in \mathcal{I}$.

Examples of matroids include:

- The colorful matroid[1], in which each element $e \in U$ has a color, and a set of elements $S \subseteq U$ is *independent* (i.e. $S \in \mathcal{I}$) if and only if there are no duplicates colors in S.
- The graphic matroid, where the elements U correspond to the edges of a graph $G = (V, E)$, and a set of elements (edges) $S \subseteq E$ is independent if and only if it there are no cycles in S.

For $X \subseteq U$, the *rank* of X, denoted by $\text{rank}(X)$, is the size of the largest independent set contained in X, i.e., $\text{rank}(X) = \max_{S \in \mathcal{I}} |X \cap S|$. For example, in the colorful matroid, the rank counts the number of distinct colors inside X. The *rank* of a matroid $\mathcal{M} = (U, \mathcal{I})$ is the rank of U.

Optimization Problems. We now define the problems we study in this chapter. We begin with the matroid intersection problem.

Matroid Intersection.
Given two matroids $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$ over the same ground set U, find a *common independent set* $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum size.

We use $n \coloneqq |U|$ to denote the ground set size and r to denote the rank of the input matroids. Perhaps the simplest and most well-known example of the matroid intersection problem is the *bipartite maximum matching* problem (discussed in Chapter 4), in which the ground set are the edges, and the two input matroids are both colorful matroids with the colors corresponding to the vertices that edges are incident to in the two sides of the graphs. In this case, $n = |E|$ and $r = |V|$. Another matroid intersection problem is the *colorful spanning tree* problem, in which, given a (edge-)colored graph, one should find a spanning tree without duplicate colors among its edges.

Further, we look at the closely related matroid union problem.

Matroid Union.
Given k matroids $\mathcal{M}_1 = (U, \mathcal{I}_1), \ldots, \mathcal{M}_k = (U, \mathcal{I}_k)$ over the same ground set U, find k independent sets $S_i \in \mathcal{I}_i$ such that their union $S_1 \cup S_2 \cup \cdots \cup S_k$ has maximum possible size.

Indeed, matroid union and matroid intersection reduce to each other in polynomial time, and algorithms for the two problems are very similar. As a concrete example of matroid union, consider the *k-disjoint spanning tree* problem. In this problem, we are given a graph $G = (V, E)$, and are asked to find k edge-disjoint

[1] A special case of the partition matroid.

spanning trees (or determine if it is impossible). The ground set U would correspond to the edge set E, and the k input matroids would all be identically set as the graphic matroid of G.

There are many other problems which can be reduced to either matroid intersection or matroid union, see Table 3.1.

Oracle Access. Since a matroid, in general, requires exponentially many bits to specify, algorithms for matroid intersection and union are commonly not given an explicit representation of the input matroids. Instead, the algorithms access the matroid by asking queries to oracles. The most well-studied query model is the *independence query*, in which one can specify a subset $S \subseteq U$ and ask if it is independent or not (i.e. "Is $S \in \mathcal{I}$?"). Another well-studied model is the stronger *rank query* model, where the oracle instead answers with the rank of S, i.e., $\text{rank}(S)$. Note that the rank query is strictly more powerful, since, by definition, $\text{rank}(S) = |S|$ if and only if $S \in \mathcal{I}$.

The traditional efficiency measure for matroid intersection algorithms is the number of queries it uses. Section 3.2 (and Paper E [BMNT23]) defines an alternative cost measure—called *dynamic oracle model*—based on how matroid intersection is often used in applications. The intuition is that a query that is "far" from previous queries should be more expensive than a query that is "close" (as "close" queries would be cheaper to process if the oracles were implemented by a dynamic data structure).

3.1 Contribution

We push the augmenting path based framework for matroid intersection and union further, which lets us obtain several new results. Highlights include (i) the first matroid intersection algorithm using a subquadratic number of independence queries, (ii) the first linear-"in-dense-matroids" rank-query algorithm for matroid union, (iii) the first sublinear depth parallel matroid intersection algorithms, and (iv) the first super-linear query lower bound. We describe these contributions below.

Subquadratic Matroid Intersection. In the independence oracle model, beating the $\tilde{O}(n^2)$ bound, known as the quadratic barrier, was an open problem that captures the limits of techniques from two lines of work. The first one is the classic augmenting-paths based algorithm of [Cun86], whose $\tilde{O}(n^2)$-query implementations were shown by [CLSSW19; Ngu19]. The other one is the continuous optimization based approach of [LSW15]. In our work, we break this "quadratic barrier" and advance the augmenting path framework for matroid intersection.

Theorem 3.1.1. *Matroid intersection can be solved using either:*

- $\tilde{O}(nr^{3/4})$ *independence queries by a randomized algorithm, w.h.p.,*

- $\tilde{O}(n^{11/6})$ *independence queries by a deterministic algorithm, or*
- $\tilde{O}(n\sqrt{r})$ *rank queries by a deterministic algorithm.*

The above algorithms also work in the dynamic oracle *models.*

There are two key ingredients to the above theorem: a subquadratic way of even finding a single augmenting path, and an efficient approximation algorithm. The rank query algorithm was shown by [CLSSW19], and in Paper E [BMNT23], we generalize this to the *dynamic* rank oracle model (as described later in Section 3.2). In the independence query algorithms, a routine to find a single augmenting path fast is given in Paper B [BBMN21] and an improved approximation algorithm in Paper C [Bli21].

Linear Matroid Union in Dense Matroids. Known reductions between matroid intersection and matroid union of k martroids implies a $\tilde{O}_k(n\sqrt{r})$ rank-query[2] matroid union algorithm based on Theorem 3.1.1. In Paper E [BMNT23], we give an improved algorithm when $r = o(n)$, using $\tilde{O}_k(n + r\sqrt{r})$ rank queries. In many graph problems, this would translate to an $\tilde{O}(|E| + |V|\sqrt{|V|})$ running time—that is linear in reasonably dense graphs—as opposed to the previous bound that gives $\tilde{O}(|E|\sqrt{|V|})$.

Theorem 3.1.2. *Matroid union can be solved using $\tilde{O}_k(n + r\sqrt{r})$ rank queries. The algorithm also works in the dynamic oracle model.*

As a direct consequence, we get improved algorithms for many other problems, see also Table 3.1. Importantly, we obtain the first[3] near-linear time algorithm in reasonably dense graphs for finding two disjoint spanning trees in a graph, improving on the $\tilde{O}(|V|\sqrt{|E|})$ bound of [GW88; Gab91].

Corollary 3.1.3. *Two disjoint spanning trees in a graph $G = (V, E)$ can be found in $\tilde{O}(|E| + |V|\sqrt{|V|})$ time.*

Sublinear Parallel Matroid Intersection. Despite recent advancements in sequential matroid intersection algorithms, the fastest poly(n)-query parallel algorithm remained a trivial $O(n)$-round parallel implementation of Edmonds' method from the 1960s. [CCK21] established a lower bound of $\Omega(n^{1/3})$ rounds for any parallel rank-query algorithm, questioning if this lower bound could be improved to $\Omega(n)$ or if sublinear-round algorithms were possible.

In Paper D [Bli22], we resolve this open problem by presenting the first sublinear-round parallel matroid intersection algorithms, breaking the trivial $O(n)$ bound in both rank-oracle and independence-oracle models.

[2]Where $O_k(\cdot)$ also hides poly(k) factors.

[3]Independently and concurrently to our work, [Qua23] achieved similar query bounds for matroid union and running time for the k-disjoint spanning tree problem.

Theorem 3.1.4. *Matroid intersection can be solved in parallel using either:*

- $O(n^{3/4})$*-rounds of rank queries, or*
- $O(n^{7/8})$*-rounds of indpendence queries.*

Superlinear Lower Bound. We show the first super-linear independence query lower bounds for matroid intersection and union problem.[4] Our simple lower bound improves on the previous $\approx 1.58n$ bound by [Har08a] and answers an open problem raised by, e.g., [Wel76] and [CLSSW19].

Theorem 3.1.5. *Any deterministic algorithm requires* $\Omega(n \log n)$ *independence queries to solve matroid union or matroid intersection.*

The lower bound is via a reduction to (s, t)-reachability in directed graphs in the communication complexity setting. We refer to Section E.8 in Paper E [BMNT23] for more details on the lower bound.

3.2 Dynamic Oracle & Applications

As alluded to previously, there are many combinatorial optimization problems that reduce to matroid intersection or matroid union. Designing algorithms for the matroid problems is thus a good way of obtaining a unified approach to solve multiple problems simultaneously.

However, in the traditional oracle models, there is one issue with the above approach: the reductions do not necessarily allow for *efficient* algorithms. Indeed, even if the holy-grail of getting a near-linear query matroid intersection/union algorithm was achieved, *this would not imply near-linear time algorithms for other problems.* For example, consider solving the two-disjoint spanning tree problem using matroid intersection. Each issued query S about a set of edges in the graph needs $O(|S|)$ time to even specify the query S, let alone answer the query (which would involve computing the number of connected components induced by S in the graph). Due of this, previous works (see e.g., [GT79; RT85; GS85; GW88; FS89; GX89; Gab91; XG94]) designed algorithms for specific problems by simulating matroid intersection/union algorithms and coming up with clever ideas to speed up the simulation for each of these problems individually.

This motivates a new oracle model—which we call *dynamic oracle*—that captures how matroid algorithms are used in reductions from other problems. This is our main conceptual contribution in Paper E [BMNT23].

[4] The lower bound also holds in the dynamic-rank-oracle model.

Problem	Our Bounds	State-of-the-Art Bounds
Via matroid union:		
k-forest	$\tilde{O}(\|E\| + (k\|V\|)^{3/2})$ ✓	$\tilde{O}(k^{3/2}\|V\|\sqrt{\|E\|})$ [GW88]
k-pseudoforest	$\tilde{O}(\|E\| + (k\|V\|)^{3/2})$ ✗	$\|E\|^{1+o(1)}$ [CKLPGS22]
k-disjoint spanning trees	$\tilde{O}(\|E\| + (k\|V\|)^{3/2})$ ✓	$\tilde{O}(k^{3/2}\|V\|\sqrt{\|E\|})$ [GW88]
arboricity	$\tilde{O}(\|E\|\|V\|)$ ✗	$\tilde{O}(\|E\|^{3/2})$ [Gab95]
tree packing	$\tilde{O}(\|E\|^{3/2})$	$\tilde{O}(\|E\|^{3/2})$ [GW88]
Shannon Switching Game	$\tilde{O}(\|E\| + \|V\|^{3/2})$ ✓	$\tilde{O}(\|V\|\sqrt{\|E\|})$ [GW88]
graph k-irreducibility	$\tilde{O}(\|E\| + (k\|V\|)^{3/2} + k^2\|V\|)$ ✓	$\tilde{O}(k^{3/2}\|V\|\sqrt{\|E\|})$ [GW88]
(f,p)-mixed forest-pseudoforest	$\tilde{O}_{f,p}(\|E\| + \|V\|\sqrt{\|V\|})$ ✓	$\tilde{O}_{f,p}(\|V\|\sqrt{f\|E\|})$ [GW88]
Via matroid intersection:		
bipartite matching	$\tilde{O}(\|E\|\sqrt{\|V\|})$ ✗	$\|E\|^{1+o(1)}$ [CKLPGS22]
graphic matroid intersection	$\tilde{O}(\|E\|\sqrt{\|V\|})$	$\tilde{O}(\|E\|\sqrt{\|V\|})$ [GX89]
simple job scheduling matroid intersection	$\tilde{O}(n\sqrt{r})$	$\tilde{O}(n\sqrt{r})$ [XG94]
convex transversal matroid intersection	$\tilde{O}(\|V\|\sqrt{\mu})$	$\tilde{O}(\|V\|\sqrt{\mu})$ [XG94]
linear matroid intersection	$\tilde{O}(n^{2.529}\sqrt{r})$ ✗	$\tilde{O}(nr^{\omega-1})$ [Har09]
colorful spanning tree	$\tilde{O}(\|E\|\sqrt{\|V\|})$	$\tilde{O}(\|E\|\sqrt{\|V\|})$ [GS85]
maximum forest with deadlines	$\tilde{O}(\|E\|\sqrt{\|V\|})$ ✓	(no prior work)

Table 3.1: Implications of our matroid algorithms from Paper E [BMNT23] in comparison with previous results. Results marked with a ✓ improve over the previous ones. Results marked with a ✗ are worse than the best time bounds. Other results match the currently best-known algorithms up to poly-logarithmic factors.

Dynamic Oracle. For a matroid $\mathcal{M} = (U, \mathcal{I})$, the i'th query $S_i \subseteq U$ is only allowed to be modified from some previous query S_j $(j < i)$ by either inserting or removing exactly one element (with the convention that $S_0 = \emptyset$ is the "0th" query). Equivalently, if the algorithm wants to query a set S, the cost of this query is how far (size of symmetric difference) it is from *any* previous query it has issued.

The intuition behind this definition is that for many instances of matroids, there already exists dynamic data structures[5] that efficiently maintain, for example, the rank of a set S undergoing insertions or deletions. Consider the graphic matroid, where one can employ dynamic connectivity data structures to implement each dynamic rank query in randomized $O(\text{polylog}n)$ [KKM13; GKKT15] or deterministic $n^{o(1)}$ [CGLNPS20; NSW17] time.

Unlike the traditional oracle models, future improved matroid algorithms in the dynamic oracle model would directly imply improved bounds for many specific problems in a unified way.

In Paper E [BMNT23], we present new techniques which allow us to implement the state-of-the-art matroid intersection algorithms from [CLSSW19], Paper B [BBMN21], and Paper C [Bli21] in the dynamic rank and dynamic independence oracle models instead (Theorem 3.1.1). Furthermore, our algorithm can be adapted for matroid union with minimal modifications. By additionally leveraging the specific structure of matroid union problems, we achieve improved bounds when r is significantly smaller than n (Theorem 3.1.2).

[5] It is okay for the data structures to be randomized and only work against oblivious adversaries, however, we require worst-case update time (as opposed to amortized).

Our approach allows for a problem-unified algorithm applicable to various problems by integrating existing dynamic data structures in a black-box way. This method either matches or surpasses the previously known best running times for many of these problems, see Table 3.1.

3.3 Augmenting Paths

Augmenting paths are central to many matroid intersection algorithms, including all of ours. While alternative approaches exist, such as convex optimization techniques (e.g., [LSW15]) and fast matrix multiplication (e.g., [Har08b]), these have not proven as effective for matroid intersection as they have for the related maximum flow and bipartite matching problems (e.g., [LS14; BLNPSSSW20; CKLPGS22]). One significant challenge is that the linear program representation of a matroid requires an exponential number of constraints, making it difficult to achieve high efficiency through continuous optimization. Consequently, the augmenting paths framework remains the most promising approach for developing fast matroid intersection algorithms. Although this framework shares similarities with its application in maximum flow and bipartite matching problems, matroid intersection presents unique challenges that increase its complexity.

The augmenting paths framework for matroid intersection (and union) originates from the work of Edmonds in the 1960s, (giving the first polynomial query algorithms) and have since been developed in a long line of research (e.g., [EDVJ68; Edm70; AD71; Law75; Edm79; Cun86; LSW15; Ngu19; CLSSW19; Qua23; Qua24]). In 1986, Cunningham [Cun86] designed an algorithm with query complexity $O(nr^{1.5})$ based on the "blocking flow" ideas akin to Hopcroft-Karp's [HK73] and Dinic's [Din70] algorithms for bipartite matching and maximum flow. For nearly three decades, this was the most efficient algorithm until the recent works of [CLSSW19; Ngu19] who independently showed that Cunningham's algorithm can be implemented using only $\tilde{O}(nr)$ independence queries.[6] In general, r can be as large as n, making this term $\tilde{O}(n^2)$, and was known as the *quadratic barrier* [CLSSW19]. A natural question is whether this barrier can be broken [LSW15, Conjecture 13], which we resolved in Theorem 3.1.1 (Paper B [BBMN21]).

Prior to our work, [CLSSW19] showed that subquadratic algorithms are possible using the stronger rank oracle (as opposed to the more classic independence oracle) by a rather simple $\tilde{O}(n\sqrt{r})$-rank-query "blocking-flow" algorithm. Moreover, [CLSSW19] also showed a subquadratic $(1-\varepsilon)$-approximation algorithm using $O(\frac{n\sqrt{n}}{\varepsilon\sqrt{\varepsilon}})$ independence queries.

Exchange Graph. The augmenting path algorithms work by (implicitly) constructing the so-called *exchange graph*. Given a matroid intersection problem over

[6] In a separate line of work based continuous optimization methods, [LSW15] showed a cutting plane algorithm using $\tilde{O}(n^2)$ independence queries.

$\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$, and a common independent set $S \subseteq \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph with a vertex for each element in the ground set, plus two additional vertices: a source s and a sink t. The edges of the graph represent valid "exchanges" in the respective matroids, for example, $S - a + b \in \mathcal{I}_1$ implies a directed edge from a to b (see Section B.2 for a formal and complete definition). Finding a shortest (s, t)-path in this graph allow one to "augment" the set S to another set S' of size $|S'| = |S| + 1$. On the other hand, if no such path exists, it serves as a certificate that S is a maximum common independent set (and hence, an answer to the matroid intersection problem).

3.3.1 Challenges

There are a few unique challenges to designing efficient (in particular, subquadratic or even $o(nr)$) algorithms using the augmenting paths approach for matroid intersection that do not appear for augmenting paths algorithms for maximum flow or bipartite matching.

Single Augmenting Path. The first challenge is that while the exchange graph $G(S)$ only has $O(n)$ vertices, it can have up to $\Omega(nr)$ edges, each taking a query to determine whether it is in the graph or not. Hence, even constructing the graph $G(S)$ explicitly is too expensive. To overcome this challenge, the algorithms cannot build the full exchange graph but must instead query and explore only the relevant parts of the graph.

Even finding a single augmenting path is thus not easy, and prior to our work, the best algorithm took $\Omega(nr)$ time to do so. In Paper B [BBMN21], we overcome this by showing a randomized $\tilde{O}(n\sqrt{r})$ and deterministic $\tilde{O}(nr^{2/3})$ independence query algorithm. We isolate a graph reachability problem with specific query access, and solve it in few phases where in each phase, we find a new "heavy" vertex reachable from the source which guarantees that we make sufficient progress. See Paper B [BBMN21] for more details.

Simultaneous Augmenting Paths. Starting with some common independent set S, one needs to find up to r augmenting paths before it has maximum size. After finding an augmenting path in $G(S)$, which lets us obtain S' of size larger than S, we now need to search for augmenting paths in the new exchange graph $G(S')$ instead. In the maximum flow problem, after finding and augmenting along a path, the residual graph is updated by reversing this path (recall Figure 1.3 on page 9). Similarly, edges are also reversed in the exchange graph, however the complications do not end here. There might be edges that existed in $G(S)$ that no longer exist in $G(S')$, and vice versa, even between vertices that where not touching the augmenting path. Hence, the exchange graph will, after each augmentation, change in rather unpredictable ways. This makes it difficult to reuse any knowledge on how the exchange graph looks like after an augmentation. Even if we could

find each augmenting path in $O(n)$ independence queries (which we do not know how to do), this would only give a $O(nr)$ bound for iteratively finding the up to r augmenting paths one by one.

Instead, one can try to find multiple augmenting paths at once, and potentially use less time when the paths are short (similar to a "blocking-flow" approach in bipartite matching or maximum flow). This is still quite challenging, since, unlike the case for maximum flow and bipartite matching, a set of (vertex) disjoint augmenting paths might not be compatible with each other (as explained above, augmenting along one of these paths might remove an edge of another path, making it invalid). In [CLSSW19], they introduce a notion of *augmenting sets*, which corresponds to a set of "compatible" augmenting paths that can be augmented along simultaneously, i.e., akin to a "blocking flow". They also give an algorithm computing these blocking flows, and after eliminating all paths in the exchange graph of length at most $\approx \frac{1}{\varepsilon}$ using $O(n^{1.5}/\varepsilon^{1.5})$ independence queries they have identified a $(1-\varepsilon)$ approximation. In Paper C [Bli21], we improve this approximation algorithm with new ideas, resulting in a $O(n\sqrt{r}/\varepsilon)$ time bound, also improving the exact algorithm to $o(nr)$.

3.3.2 Subquadratic Algorithm

To achieve our $o(nr)$ independence query matroid intersection algorithm of Theorem 3.1.1, we use two phases:[7]

- **Many Short Augmenting Paths.** We first use our Paper C [Bli21] (Algorithm C.6) to find a $(1-\varepsilon)$-approximation, i.e., a set S such that it is at most εr away from optimal.
- **Few Long Augmenting Paths.** For the remaining εr augmenting paths, we find them one by one using our graph reachability algorithm of Paper B [BBMN21] (Algorithm B.2).

Combined, we get the subquadratic $O(nr^{3/4})$ randomized or $O(n^{11/6})$ deterministic independence-query algorithm.

3.3.3 Parallel Algorithm

A parallel matroid intersection algorithm accesses the oracle in rounds. In each round, a polynomial number of queries—that may only depend on the answers to queries made in previous rounds—can be issued in parallel. The goal is to use as few rounds (called the *depth* or *adaptivity*) as possible.

Often, in parallel algorithms, the goal is polylog(n) depth. This is achievable for important special cases of matroid intersection, such as bipartite matching and linear matroid intersection with randomized algorithms [Lov79], but it remains

[7] For technical reasons, the actual algorithms need an additional third phase in the middle to handle the "reasonably many", "reasonably long" paths. We refer to Algorithms B.3 and C.7 for details.

an important open research question if this can be made deterministic, see e.g., [FGT21; GT20] for progress. However, for general matroid intersection there are lower bounds asserting that $\tilde{\Omega}(n^{1/3})$ independence query depth [KUW85] or rank query depth [CCK21] are necessary, even for randomized algorithms.[8] A natural question is "if the lower bound can be improved to $\tilde{\Omega}(n)$, or if there can be $o(n)$-round poly(n)-query algorithms" [CCK21]?

We resolve this question in Paper D [Bli22]. Our sublinear depth parallel algorithm of Theorem 3.1.4 follows a similar approach as the sequential ones. In the parallel setting, finding a single augmenting path can be done in a single round by constructing the full exchange graph. This leads to a simple $O(r)$-round algorithm of finding augmenting paths one by one. If we want to get sublinear depth, we need to find several augmenting paths at once in parallel. This is very similar to the challenge of finding many augmenting paths simultaneously, which was needed also in the sequential setting. Indeed, the "blocking-flow" approximation algorithm of [CLSSW19] can be adapted to work in the parallel setting, and this is what we accomplish in Paper D [Bli22], giving a $O(\sqrt{n}/\varepsilon)$ rounds of rank queries or $O(n^{3/4}/\varepsilon)$ rounds of independence queries parallel algorithm. Falling back to finding the remaining εr augmenting paths one by one (and setting ε appropriately) gives our sublinear algorithms using a depth of $O(n^{3/4})$ (rank) or $O(n^{7/8})$ (independence).

[8] The independence query lower bound even holds for finding a basis—a maximal independent set—of a single matroid; in contrast, this can be done with a single round of rank queries.

Chapter 4

Matchings

This chapter explores two problems about matchings—*bipartite maximum matching* and *edge coloring*—and overviews our results from Paper F [BBEMN22], Paper G [BK23], Paper H [BSVW24b], and Paper I [BSVW24a].

Matching. A matching M of a graph $G = (V, E)$ is a subset of edges in which no two edges of M share the same vertex. They play a central role in computer science and discrete mathematics, and has been studied from many perspectives. Matchings often serve as a benchmark for how well our graph techniques can be applied in various models of computation, including sequential, streaming, dynamic, online, parallel, distributed, communication, and query settings (see e.g., [HK73; Lov79; KUW85; KVV90; MS04; Zha04; IKLSW12; Mad13; Mad16; GO16; GG17; BHR19; DNO19; AV20; AK20; JST20; BLNPSSSW20; Nis21; AR20; CKPSSY21; FGT21; AB21b; ALT21; FGLPSY21; RSW22; CK24b; Waj20], and many more).

Optimization Problems. We now present the two optimization problems related to matchings that are considered in this thesis. The first one is that of finding a maximum cardinality matching in *bipartite* graphs—that is, graphs whose vertices can be divided into two sets L and R so that all edges have exactly one endpoint in L and the other in R.

Bipartite Maximum Matching.
Given a bipartite graph $G = (V, E)$, find a matching M of maximum size.

Bipartite maximum matching—or *bipartite matching* for short—is a special case of both the maximum flow and matroid intersection problems, unlike the closely related problem of finding matchings in general graphs. Maximum matching on general graphs is another interesting graph problem, however, this thesis focuses on the bipartite case.

We also study a different fundamental graph problem about matchings, namely that of covering all the edges of the graph with the fewest amount of matchings.[1] An equivalent formulation represents each matching with a color, and the goal is to color the edges with few colors so that no two incident edges get the same color. Edge coloring has applications in, for example, some scheduling problems: consider a group of people represented by vertices, with some two-persons tasks represented by edges; an edge coloring of the graph is a schedule where each color represents a time slot where simultanuous tasks can be performed.

Edge Coloring.
An edge coloring of a graph $G = (V, E)$ is an assignment of colors to the edges so that no two edges of the same color share a vertex. The *edge coloring problem* asks to color the edges of the graph using a fewest number of distinct colors possible.

Throughout this chapter, we let $n := |V|$, $m := |E|$, and $\Delta =$ maximum degree (an important parameter in edge coloring).

Connections with Maximum Flow and Matroid Intersection. We note that the bipartite maximum matching problem can be solved by our maximum flow and matroid intersection algorithms from Chapters 2 and 3. In particular, the fastest matroid intersection algorithm we achieved (in the dynamic rank oracle setting) solves bipartite matching in $\tilde{O}(m\sqrt{n})$ time. Not surprisingly, this matches (up to log-factors) the running time of Hopcroft-Karp [HK73] and Dinitz [Din70] blocking flow algorithms—indeed the matroid intersection algorithms are inspired by these.

The $O(m\sqrt{n})$ algorithm [HK73; Din70], together with an $O(n^\omega)$ algorithm[2] [IM81] for dense graphs, remained the fastest bipartite matching algorithms for a long time, until [Mad13] introduced a new approach based on continuous optimization methods. As discussed in Chapter 2, this line of continuous optimization methods research recently culminated in the almost linear time $m^{1+o(1)}$ algorithm of [CKLPGS22].

As in the case for maximum flow, the fastest *combinatorial* or *augmenting-path*-based algorithms still remained the $O(m\sqrt{n})$ algorithms [HK73; Din70] for more than 50 years. In their exciting and recent work, [CK24a; CK24b] showed the first improvement in *combinatorial* algorithms for bipartite matching—namely $O(m^{1/3}n^{5/4})$ and subsequently $n^{2+o(1)}$ running time, almost linear time in dense graphs. Our combinatorial $n^{2+o(1)}$ maximum flow algorithm (Chapter 2 and Paper A [BBST24]) was developed independently with different techniques, and achieves the same running time bound as the one in [CK24b] when run on the special case of bipartite matching. See more details in Section 2.1.

[1] Recall the similar *arboricity* problem—covering all edges of the graph with the fewest number of spanning trees—which can be stated as a *matroid union* problem; see also Chapter 3.

[2] The parameter $\omega \approx 2.371$ is the exponent of the fastest known matrix multiplication algorithm.

4.1 Models of Computation

Computing power has become increasingly fast and cheap, and is often not the limiting factor. Instead, a key challenge lies in how to effectively distribute and parallelize computations across multiple devices, as well as efficiently managing the communication channels between these devices. Theoretical models like *communication complexity*, and *parallel algorithms* capture these system-level constraints.

Query complexity and *sublinear algorithms* model the scenario where it is too expensive to store all the input[3] in memory, so algorithms can potentially save resources by querying (for example by reading from a large data base) only relevant parts of the data.

Other important models are the *dynamic* and *online* models, which address the common scenario where input data is not static, but changes over time. As an example, consider a navigation app: roads might be closed or reopened due to traffic and/or construction. Recomputing the entire solution from scratch after each small update would be inefficient and wasteful of resources. In the dynamic model, the goal is to design efficient algorithms that can quickly update the solution in response to changes in the input. In the online model, irrevocable decisions must be taken after each update before seeing the full input, making it challenging to construct near-optimal solutions.

A unifying theme across these models is to design algorithms that work only with partial access to the input data. We repeat Question 3 below.

> **Question 3.** *Under what resource limitations are our combinatorial optimization problems still efficiently solvable; specifically, can we develop matching algorithms that work in modern models of computation?*

Contributions. We show novel algorithms and insights for the bipartite matching and edge coloring problem in various models of computation. Specifically:

- in Section 4.2 we essentially settle the *communication* and various *query complexities* of bipartite maximum matching;
- in Section 4.3 we show a *dynamic* algorithm which maintains an almost maximum matching in a bipartite graph undergoing edge insertions in constant[4] update time; and
- in Section 4.4 we show *online* algorithms that edge-colors graphs, undergoing edge or vertex arrivals, using almost optimal number of colors, essentially settling the online edge coloring problem.

All these algorithms, and their analyses, turn out to be rather simple.

[3] Like in the matroid intersection and union problems (Chapter 3), where it takes exponential space to store the matroids explicitly, and the design of algorithms assumes query access instead.

[4] $O(\text{poly}(1/\varepsilon))$ update time for $(1-\varepsilon)$-approximation.

4.2 Communication and Query Complexity

In Paper F [BBEMN22], we resolved the computational complexities of bipartite matching across multiple models. Our work provides tight bounds, up to poly-logarithmic factors, in at least five computational settings: two-party communication, AND query, OR query, XOR query, and quantum edge query models, summarized in Table 4.1. These new results resolve longstanding open problems raised in, e.g., [HMT88; IKLSW12; DNO19; Nis21], and tighten the lower bounds of [BN21; Zha04].

To achieve these results, we employ simple applications of known techniques: cutting-planes methods for the new upper bounds, and reductions from set disjointedness and equality for the new lower bounds.

Moreover, the techniques almost seamlessly extend to the more general maximum-cost bipartite b-matching problem, and hence, by standard reductions, also to many other problems including, for example, transshipment and negative weight single source shortest paths.

We begin by defining the models of computation we considered.

Model	Previous papers		**Our Results**
	Lower bounds	Upper bounds	
Communication	$\Omega(n)$ Rand, $\Omega(n \log n)$ Det	$\tilde{O}(n^{1.5})$ [DNO19; IKLSW12]	$O(n \log^2 n)$, Det
Quantum edge query	$\Omega(n^{1.5})$ [Zha04; Ben22]	$O(n^{1.75})$ [LL15]	$\tilde{O}(n^{1.5})$
OR-query	$\Omega(n)$ Rand, $\Omega(n \log n)$ Det, [BN21]	$\tilde{O}(n^{1.5})$ Det, [Nis21]	$O(n \log^2 n)$, Det
XOR-query	$\Omega(n)$ Rand $\Omega(n^2)$ Det [BN21]	$\tilde{O}(n^{1.5})$ Rand [Nis21]	$O(n \log^2 n)$, Rand
AND-query	$\Omega(n)$ Rand, $\Omega(n^2)$ Det [BN21]	$O(n^2)$ Trivial	$\Omega(n^2)$, Rand

Table 4.1: The communication and query complexity bounds for bipartite matching.

Communication. In the two-party communication setting, the edges of a graph $G = (V, E)$ are distributed between two players Alice and Bob. That is, Alice has a set E_A and Bob a set E_B so that $E = E_A \cup E_B$. Alice and Bob proceed to interact with each other: in each round either Alice or Bob sends a few bits of information to the other player. The goal is to solve the problem (in our case bipartite matching on G) using as few bits of communication as possible—called

the communication complexity. Communication complexity models scenarios where the bottleneck is the data transfer, and are often useful to prove unconditional information theoretic lower bounds for other models of computation. Bipartite matching has been extensively studied in the communication setting, see e.g., [BFS86; HMT88; IKLSW12; DNO19; FKMSZ05; GKK12; GO16; AKL17; AB19a; AR20; GO16; HRVZ15; AKLY16; Kap21; KMT21; HRVZ20; Kap21; KKS14; KKS14; KMNT20; AB21b; Ben22; BN21; Rot82; Ten02; Ber09; HMT88; BFS86; IKLSW12; Raz92; IKLSW12; IKLSW12; DNO19; Nis21].

A trivial approach for the bipartite matching problem would be for Alice to send all her edges to Bob, which would require $O(m \log n)$ or $O(n^2)$ bits of communication. [IKLSW12; DNO19; Nis21] showed that $O(n\sqrt{n} \log n)$ bits suffice (by essentially simulating the sequential $O(m\sqrt{n})$ Hopcroft-Karp [HK73] blocking flow algorithm), while [HMT88] showed that $\Omega(n \log n)$ bits are needed. Our result, as seen in Table 4.1, is that $O(n \log^2 n)$ bits are sufficient, only a single log-factor away from the known lower bound.

Query Complexity. In the query models, the edges of the underlying graph $G = (V, E)$ is unknown. Algorithms must instead ask queries about the graph, and the challenge is to do so selectively to solve the problem without learning the entire graph. In the standard edge query, one can ask if $(u, v) \in E$ or not. In the remaining queries, one can specify a set S of pairs of vertices, and ask if

- OR-query: any pair in S is an edge, i.e. "is $|S \cap E| > 0$?"
- AND-query: all pairs in S are edges, i.e. "is $|S \cap E| = |S|$?"
- XOR-query: an odd number of pairs in S are edges, i.e. "is $|S \cap E|$ odd?"

We also consider the quantum edge query, and independent set query (see more details in Paper F [BBEMN22]). In a given query model, the goal is to solve the graph problem with as few queries as possible. A trivial algorithm can find the underlying graph by asking $O(n^2)$ queries—one for each potential edge. As seen in Table 4.1, our results is that $O(n \log^2 n)$ OR-queries or randomized XOR-queries is enough, while we also give simple lower bounds showing that $\Omega(n^2)$ AND-queries or deterministic XOR-queries are needed. Our algorithm also works in the quantum setting, where it uses $\tilde{O}(n\sqrt{n})$ quantum edge queries, improving on the algorithm of [LL15] and matching the lower bound of [Zha04; Ben22] up to log-factors.

Sketch of Techniques. The most interesting part of out paper are the new upper bounds. Our approach is surprisingly simple, and the same unified framework works for giving (essentially) tight upper bounds for communication complexity, OR-queries, randomized XOR-queries, and quantum edge queries. For a bipartite graph $G = (V, E)$ with $V = L \cup R$, $|L| = |R| = n$, consider the linear program for maximum matching.

$$\begin{array}{lll} \text{maximize} & \sum_{e \in E} y_e & \\ \text{subject to} & \sum_{e \in E, |e \cap \{u\}|=1} y_e \leq 1 & \forall u \in V \\ & 0 \leq y_e \leq 1 & \forall e \in E \end{array} \tag{P}$$

It is well known that the linear program for bipartite matching is integral (see e.g, [KVKV11, Section 5]). It is more advantageous to look at the dual problem: minimum vertex cover. The dual has two advantages: it has lower dimension (n instead of m), and each constraint corresponds to a single edge (so they are, for example, cheap to communicate).

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \forall (u,v) \in E \\ & 0 \leq x_v \leq 1 & \forall v \in V \end{array} \tag{D}$$

For simplicity, say we want to determine if G has a perfect matching or not. This is equivalent to checking if (D) has a feasible solution x with objective value $\sum x_v$ at most $n - \frac{1}{2}$. Indeed, by duality, if a perfect matching exists, (D) has optimal value n, and if no perfect matching exists, there is a feasible solution to (D) with value at most $n - 1$.

We can now employ a standard cutting-planes method. We keep track of a set of discovered edges E', initially empty. We then pick the center of gravity[5] point $x^\star$ of

$$\left\{ x \in [0,1]^V : \sum_{v \in V} x_v \leq n - \tfrac{1}{2}, \quad x_u + x_v \geq 1, \forall (u,v) \in E' \right\}. \tag{4.1}$$

A cutting plane exactly corresponds to a constraint "$x_u + x_v \geq 1$" (for some edge $(u,v) \in E$) which is violated by $x^\star$. If we can find such a violated edge, we simply add it to E' and repeat. Eventually the set in (4.1) becomes empty (in which case a perfect matching exists), or conversely we find a point $x^\star$ feasible for (D) of objective value at most $n - \frac{1}{2}$ (i.e., a fractional vertex cover serving as a certificate that no perfect matching exists).

The center-of-gravity cutting-planes method reduces the volume by a constant fraction each round. It turns out that the number of rounds necessary is then $O(n \log n)$, crucially relying on the fact that the vertex cover linear program (D) has dimension n, and our introduced slack of $\frac{1}{2}$ ("$\ldots \leq n - \frac{1}{2}$" instead of "$\ldots \leq n - 1$").

What remains is to argue how to efficiently find an violated edge in the different models of computation. That is, given a (fractional) point $x^\star$, we need to quickly

[5]This is NP-hard to compute, however, in the two-party communication setting and query settings, we do not care about the running time, only the query or communication complexity. We note that there are cutting-planes methods with analytic center [Vai89] which can be computed in polynomial time with the same volume reduction guarantee we could have used instead.

find an edge $(u, v) \in E$ such that $x^\star_u + x^\star_v < 1$ (or determine that none exists). In the two-party communication setting, this is trivial: if Alice or Bob has such an edge, they send it (using $O(\log n)$ bits of communication) to the other player. For OR-queries, we may simply do a binary search on the set of potential violated pairs $\{(u, v) : x^\star_u + x^\star_v < 1\}$ to see if any such pair exists as an edge in the graph. Randomized XOR-queries can simulate OR-queries (but interestingly, deterministic XOR-query algorithms need $\Omega(n^2)$ queries). The quantum edge query algorithm can adapt the OR-query algorithm with Grover's search to simulate each OR-queries in $O(\sqrt{n})$ quantum edge queries.

4.3 Incremental Dynamic

In the dynamic matching problem, edges are either inserted or removed from a (bipartite) graph, and the goal is to maintain a matching (preferably as close to maximum size) with low update time. A trivial algorithm recomputes the matching from scratch after each update, however the aim is to design algorithms which use much less time per update (ideally polylog(n) or even constant update time). Dynamic matching has received extensive research attention in recent years, see e.g., [GP13; BK22; Waj20; ACCSW18; BK21; BS16; PS16; AAGPS19; BFH21; CS18; NS16; San07; BHN16; BRR23].

Fully Dynamic. Unfortunately, in the fully dynamic setting (both edge insertions and deletions), there are conditional lower bounds that assert that sublinear (in n) update time seems infeasible [HKNS15] to maintain an exact maximum matching. Hence, most work focus on an approximate version where the goal is instead to maintain a relatively large matching, say factor of $\frac{1}{2}$ or a factor of $(1 - \varepsilon)$ away from optimal. A long line of research (e.g., [BGS18; BHN17; BHI18; BK19; BDHSS19; OR10; Sol16; BCH17]) led to algorithms achieving $\approx \frac{1}{2}$-approximation with polylog(n) or constant update time, and very recently it was show how to get better-than-$\frac{1}{2}$-approximation [Beh23; BKSW23] in polylog(n) update time—with the caveat that this only maintains an estimate of the *size* of the matching and not the actual approximate matching itself.

Partially Dynamic. In the $(1 - \varepsilon)$-approximation regime, achieving polylogarithmic update time for fully dynamic matching seems out of reach with current techniques and is perhaps even impossible. Hence, a recent research have focused on maintaining a $(1 - \varepsilon)$-approximate matching in partially dynamic graphs: either just for edge insertions (incremental) or just for edge deletions (decremental). For these models, poly$(\log n, 1/\varepsilon)$-update time algorithms have been shown both for incremental [Gup14; BKS23b] and decremental graphs [BPS20; JJST22; BKS23b]. The polylog(n)-dependency seem crucial in all these algorithms—which either relies on multiplicative weight updates, maximum flow computations, or continuous opti-

mization techniques—and it seems implausible that these techniques can achieve constant update time independent of n.

The first constant time (i.e., update time independent of the size of the graph) partially dynamic matching algorithm is due to [GLSSS19] who solved $(1-\varepsilon)$-approximate matching in incremental graphs with an update time of $(1/\varepsilon)^{O(1/\varepsilon)}$. Their solution relies on eliminating short ($\approx \frac{1}{\varepsilon}$-length) augmenting paths. However, their approach seems to inherently carry an exponential dependency on $1/\varepsilon$ due to the number of possible such paths present in the graph.

Our Result. To recall, the previous state of the art in incremental $(1-\varepsilon)$-approximate bipartite matching has update time of either $O(\text{polylog}(n) \cdot \text{poly}(\frac{1}{\varepsilon}))$, which depends on n, or $(1/\varepsilon)^{(1/\varepsilon)}$, which has exponential dependency on $1/\varepsilon$. In Paper G [BK23], we show a simple incremental algorithm running in constant time (no dependency on n), and only incurring polynomial dependency on $\frac{1}{\varepsilon}$.

Theorem 4.3.1. *There is an algorithm which maintains a $(1-\varepsilon)$-approximate matching in incremental bipartite graphs in amortized $O(\frac{1}{\varepsilon^6})$ update time.*

Our algorithm is based on the weighted variant of the celebrated *Edge-Degree-Constrained-Subgraph* (EDCS) matching sparsifier. First introduced by [BS15], the (unweighted) EDCS has proved a model-unified technique with applications in a number of different computational settings: streaming [Ber20; ABBMS19; AB21a], stochastic, one way communication, fault tolerant [AB19b], sub-linear [Beh23; BKSW23; BRR23; BKS23c] and dynamic [BS15; BS16; Kis22; GSSU22; Beh23]. On the other hand the *weighted* EDCS variant which provides a tighter approximation ($(1-\varepsilon)$-approximation, instead of $(\frac{2}{3}-\varepsilon)$-approximation), has only found applications in small arboricity (i.e., sparse) graphs [BS15]. Hence, our algorithm is the first to use weighted EDCS in dense graphs.

In addition to our incremental algorithm, we present new proofs for weighted EDCS sparsifier properties, including a direct identification of a fractional matching and a fractional vertex cover. This significantly simplifies the weighted EDCS approximation ratio proof of [BS15], and in addition we show that the analysis is tight. That is, we show a quadratic dependence on the slack parameter for the weighted EDCS maximum degree, contrasting with the linear relationship in unweighted EDCS [Beh21]. Our hard example likely rules out faster than $1/\varepsilon^2$ complexity algorithms using weighted EDCS in semi-streaming and distributed models, where the ε-dependency is more important than in the dynamic setting. Given the popularity and broad adoption of (unweighted) EDCS, these findings may be of independent interest.

Sketch of Techniques. As alluded to, we employ weighted EDCS as a graph sparsifier. For a graph G, the weighted β-EDCS is a multi-graph H on the same vertex set, with each edge of H corresponding to an edge in G. H must also satisfy the following properties:

(i) $\deg_H(u) + \deg_H(v) \leq \beta$ for all edges $(u, v) \in H$, and

(ii) $\deg_H(u) + \deg_H(v) \geq \beta - 1$ for all edges $(u, v) \in E$.

If $\beta = \Theta(\varepsilon^{-2})$ and H is a β-WEDCS of G then it is known that H preserves a $(1-\varepsilon)$-approximate matching. Our main contribution is showing a simple algorithm that maintains a β-WEDCS efficiently under edge insertions. This means one can efficiently maintain a $(1-\epsilon)$-approximate matching within the support of H through periodic rebuilding, similar to [GP13].

The algorithm to maintain H is straightforward: after an edge insertion (u, v) in the graph G, this new edge might violate Item (ii). If this is the case, we call the edge *underfull*, and add it (once) to the multigraph H. However, this might now mean that some neighboring edge (v, w) in H violates Item (i), since $\deg_H(v)$ just increased. Call such an edge *overfull*. If this is the case, we simply remove it (once) from H. In turn, now another edge (w, z) might be underfull again, so we need to add it and so on. This results in the algorithm following up to two greedy paths from (u, v); alternatively adding and removing edges from H.[6] Note that, except for the endpoints of these paths, the degrees in H are preserved.

This greedy algorithm turns out to be efficient, every time an underfull edge is added or an overfull edge is removed, a global "potential" goes up, guaranteeing it can only occur a total of $O(n\beta^2)$ times (the argument is similar to the ones in [AB19b; Ber20; BS15]).

There is still one remaining issue: after removing an edge from H the algorithm searches all neighboring edges to determine if there is a new underfull edges created, and this can take up to $\Omega(n)$ time if there are $\approx n$ neighbors. To fix this, we can mark vertices as "inactive" after the algorithm has scanned their neighborhoods more than β^2/ε times, and argue that in the future it is okay to ignore all the edges connected to "inactive" vertices (doing so only loses us an ε-factor of the maximum matching).

4.4 Online Edge Coloring & Rounding Fractional Matchings

The classic theorem of [Viz64] asserts that any graph of maximum degree Δ can be edge colored using at most $\Delta + 1$ colors (with Δ being a trivial lower bound). Since then, there have been a lot of algorithmic work on the edge-coloring problem.

In Paper H [BSVW24b] and Paper I [BSVW24a] we study the problem in the *online* model. In the online model, the graph is initially unknown and revealed over time, similar to the incremental dynamic model. Once an edge is revealed, the algorithm must immediately and irrevocably make a decision about this edge (i.e., assign it a color). This irrevocability is the key difference between the online

[6] This is similar to augmenting path algorithms for matching in which augmenting along a path amounts to alternating adding and removing edges on the path to the matching. Indeed, if $\beta = 2$, the definition of β-WEDCS H would correspond exactly to a *maximal* (but not maximum) matching, so the WEDCS can be seen as a generalization of maximal matchings.

and incremental dynamic models—online algorithms cannot change their mind later while incremental dynamic algorithms are allowed to. This poses a challenge for online algorithms, and it is unclear how well they can perform compared to optimal offline algorithms, even assuming unbounded computational power.

The edge-coloring problem was one of the first graph problems considered in the online model, where [BMN92] conjectured that a $(1 + o(1))\Delta$-edge-coloring can be computed online in graphs of maximum degree $\Delta = \omega(\log n)$. Conversely, if the maximum degree $\Delta = O(\log n)$, they showed that a simple greedy algorithm is optimal, using 2Δ colors.

That is, unless the the graph has very small maximum degree, the above conjecture states that online algorithms can use almost the same number of colors as the optimal (offline) Δ or $\Delta + 1$—formally the competitive ratio goes towards 1 when the size of the graph grows.

Since [BMN92], there have been a lot of progress towards resolving their conjecture for restricted settings, including random-order edge arrivals [AMSZ03; BMM12; BGW21; KLSST22] and vertex arrivals [CPW19; SW21]. In the most general setting, i.e., under adversarial edge arrivals, [KLSST22] recently provided the first algorithm outperforming the trivial 2-competitive greedy algorithm, showing an $(\frac{e}{e-1} + o(1))\Delta$-edge-coloring algorithm.

Our Results. In Paper I [BSVW24a], we resolve this longstanding conjecture in in the most general setting of adversarial edge arrivals.

Theorem 4.4.1. *There exists a randomized online algorithm that, on n-vertex graphs with known maximum degree $\Delta = \omega(\log n)$, outputs a $(1+o(1))\Delta$-edge-coloring with high probability.*

In our earlier work of Paper H [BSVW24b], we showed the same result in the more restrictive model of one-sided vertex-arrivals in bipartite graphs, improving and significantly simplifying a prior such result by [CPW19]. While the later and more general algorithm in Paper I [BSVW24a] is also simple, the algorithm and analysis in Paper H [BSVW24b] is even simpler, shorter, and achieves a better $o(1)$-term, although for a restricted arrival model.

Our algorithms also extend to generalizations of the edge coloring problem such as list edge coloring [Kah96] and local edge coloring [Chr23].

Sketch of Techniques. At the core of our most general algorithm of Theorem 4.4.1 is a new approach to round spread-out fractional matchings in the online settings. In particular, given a fractional matching y (i.e., a feasible solution to the linear programming relaxation, see (P)), we call it *ε-spread-out* for some $\varepsilon > 0$ if $y_e \leq \varepsilon^5$ for all edges e. We give an online algorithm that, when edges and their values y_e are revealed, outputs an (integral) matching M where $\Pr[e \in M] \geq (1 - \varepsilon)y_e$. That is, our online algorithm $(1 - \varepsilon)$-approximately rounds the matching, and has per-edge guarantees. This online rounding algorithm might be of independent interest.

To use this algorithm for edge coloring, we use a previously known reduction from [CPW19]. Let $y_e \coloneqq \frac{1}{\Delta}$, since this is guaranteed to be a feasible fractional matching. Then the outputted matching M of our algorithm is guaranteed to be "fair" in the following sense: each edge is chosen to be in the matching with probability at least $\frac{1}{(1+o(1))\Delta}$. Intuitively, if we select this matching M to be the first color, the remaining graph $G \setminus M$, except for a small fraction of the vertices, has maximum degree $\Delta - 1$. We can then, in an online fashion, pipeline $(1 + o(1))\Delta$ interleaved instances of the fair-matching algorithm to peel of colors one by one, and in the end only use a total of $(1 + o(1))\Delta$ colors.

Our main technical contribution is in the improved fair-matching algorithm. A natural approach is that, when edge $e = (u, v)$ arrives, we want to match it with target probability $(1 - \varepsilon)y_e$. However, since the output must be a matching, we cannot match e at all if either u or v are already matched by our algorithm. To compensate for this, one needs to scale up the probability of picking e when possible: instead pick e with probability $\frac{(1-\varepsilon)y_e}{\Pr[u,\, v \text{ both unmatched}]}$.

The goal becomes proving that quantity is indeed a probability, i.e., that it was not scaled up to much and is always at most 1. Previous approached (e.g., [CPW19; KLSST22]) analyzed versions of this algorithm trying to bound and minimize the correlations introduced.

We follow a different approach where we embrace the correlations and use a different analysis that allow them. Crucially, we present a different but still simple algorithm, with a subtle difference: instead of scaling up by $\frac{1}{\Pr[u,v \text{ both unmatched}]}$, our scaling factor depends upon the algorithm's actual execution path (sequence of random decisions) so far. This allows us to analyse the scaling factor for an edge as a *martingale* process. While there may still be correlations, we show that this martingale has (i) small step size and (ii) bounded observed variance, and hence we can get strong concentration via the powerful Freedman's inequality [Fre75]. Our change of viewpoint is crucial for achieving our result and leads to a simple and concise algorithm and analysis.

Chapter 5

Open Questions and Future Directions

This chapter discusses relevant open questions related to this thesis, and intriguing future research directions.

Linear Time Combinatorial Flow. While our new $n^{2+o(1)}$ augmenting path based algorithm achieves almost linear time in dense graphs, it does not perform as efficiently in sparse graphs. A challenging open problem is to find *augmenting path*-based or *combinatorial* maximum flow algorithms which runs in (almost) linear time also on sparse graphs, matching the continuous optimization methods achieving $m^{1+o(1)}$ time [CKLPGS22; BCKLPGSS24; BCKLMGS24].

One apparent bottleneck in our $n^{2+o(1)}$ algorithm is finding approximate maximum flows in directed acyclic graphs (DAGs). A natural question is then: *Can a combinatorial algorithm solve approximate flow on DAGs in (almost) linear time?* If this is the case, then it seems plausible that our tools (directed expander hierarchies) can generalize the solution to work on any directed graph. DAGs also seem to capture the hardness of other fundamental problems in directed graphs, such as dynamic shortest paths [BPS20; CK24b], parallel reachability [Fin18; LJS19], and diameter-reducing shortcuts [KP22]. Improving the algorithmic toolkit for DAGs is thus an important research direction.

Extending $n^{2+o(1)}$ Combinatorial Flow to Other Problems. One natural question is whether our $n^{2+o(1)}$ maximum flow algorithm can be extended to solve the more general minimum cost maximum flow problem in the same time (the continuous optimization methods can do this). A promising direction is using ideas from *length-constrained expander decomposition*, see e.g., [HHLRS24].

Other problems like maximum matching in general graphs and matroid intersection currently have no almost linear time algorithms, and it seems difficult to extend the continuous optimization methods to work for these problems. Instead, the

current state-of-the-art algorithms are based on augmenting paths: Theorem 3.1.1 for matroid intersection, and the $\tilde{O}(m\sqrt{n})$ time algorithm for general matching (see [Gab17] for an accessible version of this algorithm). In light of our $n^{2+o(1)}$ augmenting path maximum flow algorithm, a natural next step would be attempting to adapt those techniques to general matching or matroid intersection, potentially achieving linear-in-dense-{graphs/matroids} algorithms.

Settling the Complexity of Matroid Intersection and Union. There remains a gap between the lower bounds and fastest algorithms for matroid intersection. In the regime of $r = \Theta(n)$, for rank queries, the strongest lower-bound is linear, while the best algorithm uses $\tilde{O}(n^{3/2})$ queries [CLSSW19]. For independence queries, the lower-bound is $\Omega(n \log n)$ (Theorem 3.1.5) while the fastest algorithm uses $\tilde{O}(n^{7/4})$ queries (Theorem 3.1.1). This leaves a few important questions open: *Can independence-query algorithms achieve the same efficiency as rank-query algorithms, or is the rank-query substantially stronger? Are there linear query algorithms for matroid intersection or union?* If so, it would imply many faster algorithms for other problems like colorful spanning trees and two-disjoint spanning trees. Perhaps a good, but still challenging plan of attack towards resolving this question is to develop linear time algorithms for the problems of colorful spanning tree and two-disjoint spanning trees.[1] Conversely, if no linear time algorithms exists, one could ask: *Are there truly superlinear (i.e., $\Omega(n^{1+c})$ for constant $c > 0$) query lower bounds for matroid intersection?* If this is the case, it would also imply similar lower bounds for the related problem of submodular function minimization, even in the weakly-polynomial setting—currently there is an $\Omega(n \log n)$ lower bound in the strongly-polynomial setting [CGJS22]. Studying the communication complexity of matroid intersection (where Alice and Bob are each given one matroid) is a promising direction to prove lower bounds, or conversely give new insight in case a $\tilde{O}(n)$ bit protocol (matching the communication complexity of bipartite matching) is found.

Similarly, in the parallel setting, closing the gap between the current algorithms and lower bounds is interesting. The strongest lower bounds show that $\Omega(n^{1/3})$ rounds are needed [KUW85; CCK21], and some sublinear upper bounds are given in Theorem 3.1.4.

Follow-Up Work: In a recent paper (at the time of writing under review, and thus not included in this thesis) [BT25], we give simpler and lower-depth parallel algorithms than in Theorem 3.1.4, namely $O(n^{2/3})$-round of rank queries or $O(n^{5/6})$-round of independence queries. We also give the first linear-time independence query $(1-\varepsilon)$ approximation algorithm, making progress towards linear time (exact) algorithms for matroid intersection.

[1] After the original submission of this thesis, the paper [AK24] was announced, reducing k-disjoint spanning trees to k maximum flow instances—hence achieving $O(\text{poly}(k)m^{1+o(1)})$ running time—by using matroid intersection techniques specialized to this problem.

Communication Complexity of General Matching. In Section 4.4, we show tight bounds for the communication and various query complexities of bipartite matching. A natural direction is to ask about matchings on not-necessarily-bipartite graphs: *Is the communication complexity of general matching also* $\tilde{O}(n)$*?* The linear programming formulations for general matching, unlike for the bipartite case, has exponential extension complexity making it unlikely that the same cutting planes method approach would work. Thus, a positive resolution to the above question would hopefully lead to insights on the interplay between bipartite and general matching that would also prove useful outside of the communication model. Conversely, if there is a substantial difference in difficulty between bipartite and general matching, the communication setting would serve as a promising model to demonstrate this gap since it allows for information theoretical lower bounds.

Bounded Round Communication. For many graph problems, including bipartite matching, it has been shown that $\Theta(n^2)$ bits of communication is necessary if there is only a single round of communication (i.e., one-way communication) [FKMSZ05]. Our $\tilde{O}(n)$ communication protocol uses $\tilde{\Theta}(n)$ rounds. There is certainly a trade of between the number of rounds allowed and the total bits of communication needed, and studying this trade-off (for bipartite matching and other problems) is an intriguing research direction. For the bipartite matching problem, r-round protocols need at least $n^{1+\Omega(1/r)}$ total communication [AR20; GO16], ruling out $\tilde{O}(n)$ communication in, say, $\sqrt{\log n}$ rounds. A polylog(n)-round $\tilde{O}(n)$-communication protocol would be exciting if it exists, due to its connections with semi-streaming—*How many passes does an* $\tilde{O}(n)$*-space streaming algorithm need to solve bipartite matching?*—and distributed models—*How many CONGEST rounds are necessary to solve bipartite matching?* Similarly, studying the OR-query complexity for bipartite matching with bounded rounds is another interesting direction; see e.g., [ACK21] which studies graph connectivity problems in that setting.

Dynamic Matching. The big open question in dynamic matching is: *Can one maintain a* $(1-\varepsilon)$*-approximate matching in* polylog(n) *update time?* Answering this question seem far from being solved with current techniques. However, research towards this goal have led to many new graph techniques and insights. The current best update time is $m^{1/2-\Omega_\varepsilon(1)}$ by [BKS23a] with the caveat that it does not maintain the matching itself but just its size (to maintain the matching, the best known is $O(\sqrt{m})$ update time [GP13]). If only polylog(n) update time allowed, [BKSW22; Beh22] show how to maintain a slightly-better-than-$\frac{1}{2}$ approximation, but again just estimating the size. A more approachable open question is to design a constant update time (preferably poly$(1/\varepsilon)$) algorithm for *decremental* bipartite graphs, matching our result for incremental bipartite graphs in Section 4.3.

Deterministic Online Edge Coloring. Randomization is often a crucial ingredient in online algorithms (including for our online edge-coloring algorithms) and

sometimes necessary to achieve good competitive ratios. A natural question is if randomization is needed for online edge coloring, or if a deterministic version of Theorem 4.4.1 exists.

Follow-Up Work: In a recent work (at the time of writing under review, and hence not included in this thesis) [BSVW25], we make progress on this question by showing the first *deterministic* algorithm for online edge coloring in any setting to outperform greedy: a $(\frac{e}{e-1} + o(1))\Delta$-edge-coloring algorithm under one-sided vertex-arrivals for bipartite graphs, again assuming maximum degree $\Delta = \omega(\log n)$. However, it is not clear whether deterministic algorithms can beat the 2-competitive greedy algorithm in general graphs with edge arrivals, and another potential way to settle this question would be showing hardness results.

* * *

To close this thesis, we repeat the question from the introduction (Chapter 1):

Can we design efficient and simple algorithms in a unified way?

While this thesis makes progress towards this question, there still remain many problems in the field of theoretical computer science where we are yet to discover efficient, simple, and unified algorithms.

Bibliography

(Co-)Authored by Joakim Blikstad

[ABNZ24] Mikkel Abrahamsen, Joakim Blikstad, André Nusser, and Hanwen Zhang. "Minimum Star Partitions of Simple Polygons in Polynomial Time". In: *STOC*. ACM, 2024, pp. 904–910. DOI: 10.1145/3618260.3649756 (cit. on p. vi).

[BBEMN22] Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Nearly Optimal Communication and Query Complexity of Bipartite Matching". In: *FOCS*. IEEE, 2022, pp. 1174–1185. DOI: 10.1109/FOCS54457.2022.00113 (cit. on pp. vi, 11, 31, 34, 35, 333).

[BBMN21] Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Breaking the quadratic barrier for matroid intersection". In: *STOC*. ACM, 2021, pp. 421–432. DOI: 10.1145/3406325.3451092 (cit. on pp. v, 11, 21, 24, 26–29, 191).

[BBST24] Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, and Ta-Wei Tu. "Maximum Flow by Augmenting Paths in $n^{2+o(1)}$ Time". In: *FOCS*. IEEE, 2024 (cit. on pp. v, 10, 13, 15–19, 32, 69).

[BK23] Joakim Blikstad and Peter Kiss. "Incremental (1-ϵ)-Approximate Dynamic Matching in O(poly(1/ϵ)) Update Time". In: *ESA*. Vol. 274. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 22:1–22:19. DOI: 10.4230/LIPICS.ESA.2023.22 (cit. on pp. vi, 12, 31, 38, 373).

[Bli21] Joakim Blikstad. "Breaking O(nr) for Matroid Intersection". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 31:1–31:17. DOI: 10.4230/LIPICS.ICALP.2021.31 (cit. on pp. v, 11, 21, 24, 26, 29, 219).

[Bli22] Joakim Blikstad. "Sublinear-Round Parallel Matroid Intersection". In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 25:1–25:17. DOI: 10.4230/LIPICS.ICALP.2022.25 (cit. on pp. v, 11, 21, 24, 30, 243).

[BMNT23] Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, and Ta-Wei Tu. "Fast Algorithms via Dynamic-Oracle Matroids". In: *STOC*. ACM, 2023, pp. 1229–1242. DOI: 10.1145/3564246.3585219 (cit. on pp. v, 11, 21, 23–26, 267).

[BSVW24a] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. "Online Edge Coloring Is (Nearly) as Easy as Offline". In: *STOC*. ACM, 2024, pp. 36–46. DOI: 10.1145/3618260.3649741 (cit. on pp. vi, 12, 31, 39, 40, 411).

[BSVW24b] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. "Simple and Asymptotically Optimal Online Bipartite Edge Coloring". In: *SOSA*. SIAM, 2024, pp. 331–336. DOI: 10.1137/1.9781611977936.30 (cit. on pp. vi, 12, 31, 39, 40, 399).

[BSVW25] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. "Deterministic Online Bipartite Edge Coloring". In: *arXiv preprint arXiv:2408.03661* (2025). to appear in SODA 2025 (cit. on pp. vi, 46).

[BT25] Joakim Blikstad and Ta-Wei Tu. "Efficient Matroid Intersection via a Batch-Update Auction Algorithm". In: (2025). to appear in SOSA 2025 (cit. on pp. vi, 44).

Other References

[AAGPS19] Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. "Dynamic set cover: improved algorithms and lower bounds". In: *STOC*. ACM, 2019, pp. 114–125. DOI: 10.1145/3313276.3316376 (cit. on p. 37).

[AB19a] Sepehr Assadi and Aaron Bernstein. "Towards a Unified Theory of Sparsification for Matching Problems". In: *SOSA*. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 11:1–11:20 (cit. on p. 35).

[AB19b] Sepehr Assadi and Aaron Bernstein. "Towards a Unified Theory of Sparsification for Matching Problems". In: *SOSA*. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 11:1–11:20. DOI: 10.4230/OASICS.SOSA.2019.11 (cit. on pp. 38, 39).

[AB21a] Sepehr Assadi and Soheil Behnezhad. "Beating Two-Thirds For Random-Order Streaming Matching". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 19:1–19:13. DOI: 10.4230/LIPICS.ICALP.2021.19 (cit. on p. 38).

[AB21b] Sepehr Assadi and Soheil Behnezhad. "On the Robust Communication Complexity of Bipartite Matching". In: *APPROX-RANDOM*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 48:1–48:17 (cit. on pp. 31, 35).

[ABBMS19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. "Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs". In: *SODA*. SIAM, 2019, pp. 1616–1635. DOI: 10.1137/1.9781611975482.98 (cit. on p. 38).

[ACCSW18] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. "Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms". In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 7:1–7:16. DOI: 10.4230/LIPICS.ICALP.2018.7 (cit. on p. 37).

[ACK21] Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. "Graph Connectivity and Single Element Recovery via Linear and OR Queries". In: *ESA*. Vol. 204. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 7:1–7:19. DOI: 10.4230/LIPICS.ESA.2021.7 (cit. on p. 45).

[AD71] Martin Aigner and Thomas A. Dowling. "Matching Theory for Combinatorial Geometries". In: *Transactions of the American Mathematical Society* 158.1 (1971), pp. 231–245. DOI: 10.2307/1995784 (cit. on p. 27).

[AFKLM24] Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. "New Graph Decompositions and Combinatorial Boolean Matrix Multiplication Algorithms". In: *STOC*. ACM, 2024, pp. 935–943 (cit. on p. 14).

[AK20] Mohamad Ahmadi and Fabian Kuhn. "Distributed Maximum Matching Verification in CONGEST". In: *DISC*. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 37:1–37:18 (cit. on p. 31).

[AK24] Pavel Arkhipov and Vladimir Kolmogorov. *A faster algorithm for the k-forest problem: breaking the $O_k(n^{3/2})$ complexity barrier*. 2024. arXiv: 2409.20314 [cs.DS] (cit. on pp. 6, 44).

[AKL17] Sepehr Assadi, Sanjeev Khanna, and Yang Li. "On Estimating Maximum Matching Size in Graph Streams". In: *SODA*. SIAM, 2017, pp. 1723–1742 (cit. on p. 35).

[AKLY16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. "Maximum Matchings in Dynamic Graph Streams and the Simultaneous Communication Model". In: *SODA*. SIAM, 2016, pp. 1345–1364 (cit. on p. 35).

[ALPS23] Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. "All-Pairs Max-Flow is no Harder than Single-Pair Max-Flow: Gomory-Hu Trees in Almost-Linear Time". In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. IEEE, 2023, pp. 2204–2212. DOI: 10.1109/FOCS57990.2023.00137 (cit. on p. 13).

[ALT21] Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. "An Auction Algorithm for Bipartite Matching in Streaming and Massively Parallel Computation Models". In: *SOSA*. SIAM, 2021, pp. 165–171 (cit. on p. 31).

[AMSZ03] Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. "Switch scheduling via randomized edge coloring". In: *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*. 2003, pp. 502–512 (cit. on p. 40).

[AR20] Sepehr Assadi and Ran Raz. "Near-Quadratic Lower Bounds for Two-Pass Graph Streaming Algorithms". In: *FOCS*. IEEE, 2020, pp. 342–353 (cit. on pp. 31, 35, 45).

[AV20] Nima Anari and Vijay V. Vazirani. "Matching Is as Easy as the Decision Problem, in the NC Model". In: *ITCS*. Vol. 151. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 54:1–54:25 (cit. on p. 31).

[BBPNSSS22] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. "Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 20:1–20:20. DOI: 10.4230/LIPICS.ICALP.2022.20 (cit. on p. 13).

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. "Deterministic Fully Dynamic Approximate Vertex Cover and Fractional Matching in O(1) Amortized Update Time". In: *IPCO*. Vol. 10328. Lecture Notes in Computer Science. Springer, 2017, pp. 86–98. DOI: 10.1007/978-3-319-59250-3_8 (cit. on p. 37).

[BCKLMGS24] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality". In: *CoRR* abs/2407.10830 (2024). DOI: 10.48550/ARXIV.2407.10830 (cit. on p. 43).

[BCKLPGSS24] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. "Incremental Approximate Maximum Flow on Undirected Graphs in Subpolynomial Update Time". In: *SODA*. SIAM, 2024, pp. 2980–2998. DOI: 10.1137/1.9781611977912.106 (cit. on p. 43).

[BDHSS19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. "Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time". In: *FOCS*. IEEE Computer Society, 2019, pp. 382–405. DOI: 10.1109/FOCS.2019.00032 (cit. on p. 37).

[Beh21] Soheil Behnezhad. "Improved Analysis of EDCS via Gallai-Edmonds Decomposition". In: *CoRR* abs/2110.05746 (2021) (cit. on p. 38).

[Beh22] Soheil Behnezhad. "Dynamic Algorithms for Maximum Matching Size". In: *CoRR* abs/2207.07607 (2022) (cit. on p. 45).

[Beh23] Soheil Behnezhad. "Dynamic Algorithms for Maximum Matching Size". In: *SODA*. SIAM, 2023, pp. 129–162. DOI: 10.1137/1.9781611977554.CH6 (cit. on pp. 37, 38).

[Ben22] Gal Beniamini. "The Approximate Degree of Bipartite Perfect Matching". In: *CoRR* abs/2004:14318 (2022) (cit. on pp. 34, 35).

[Ber09] Dimitri P. Bertsekas. "Auction Algorithms". In: *Encyclopedia of Optimization*. Springer, 2009, pp. 128–132 (cit. on p. 35).

[Ber20] Aaron Bernstein. "Improved Bounds for Matching in Random-Order Streams". In: *ICALP*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 12:1–12:13. DOI: 10.4230/LIPICS.ICALP.2020.12 (cit. on pp. 38, 39).

[BFH21] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. "A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching". In: *ACM Trans. Algorithms* 17.4 (2021). Announced at SODA'19, 29:1–29:51. DOI: 10.1145/3469833 (cit. on p. 37).

[BFS86] László Babai, Peter Frankl, and Janos Simon. "Complexity classes in communication complexity theory (preliminary version)". In: *FOCS*. IEEE Computer Society, 1986, pp. 337–347 (cit. on p. 35).

[BGJLLPS22] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. "Faster maxflow via improved dynamic spectral vertex sparsifiers". In: *STOC*. ACM, 2022, pp. 543–556. DOI: 10.1145/3519935.3520068 (cit. on p. 13).

[BGS18] Surender Baswana, Manoj Gupta, and Sandeep Sen. "Fully Dynamic Maximal Matching in O(log n) Update Time (Corrected Version)". In: *SIAM J. Comput.* 47.3 (2018). Announced at FOCS'11, pp. 617–650. DOI: 10.1137/16M1106158 (cit. on p. 37).

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. "Online Edge Coloring Algorithms via the Nibble Method". In: *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2021, pp. 2830–2842 (cit. on p. 40).

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. "Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching". In: *SIAM J. Comput.* 47.3 (2018). Announced at SODA'15, pp. 859–887. DOI: 10.1137/140998925 (cit. on p. 37).

[BHN16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. "New deterministic approximation algorithms for fully dynamic matching". In: *STOC*. ACM, 2016, pp. 398–411. DOI: 10.1145/2897518.2897568 (cit. on p. 37).

[BHN17] 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: *SODA*. SIAM, 2017, pp. 470–489. DOI: 10.1137/1.9781611974782.30 (cit. on p. 37).

[BHR19] Aaron Bernstein, Jacob Holm, and Eva Rotenberg. "Online Bipartite Matching with Amortized $O(\log^2 n)$ Replacements". In: *J. ACM* 66.5 (2019), 37:1–37:23 (cit. on p. 31).

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. "Deterministically Maintaining a $(2+\epsilon)$-Approximate Minimum Vertex Cover in $O(1/\epsilon^2)$ Amortized Update Time". In: *SODA*. SIAM, 2019, pp. 1872–1885. DOI: 10.1137/1.9781611975482.113 (cit. on p. 37).

[BK21] Sayan Bhattacharya and Peter Kiss. "Deterministic Rounding of Dynamic Fractional Matchings". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 27:1–27:14. DOI: 10.4230/LIPICS.ICALP.2021.27 (cit. on p. 37).

[BK22] Soheil Behnezhad and Sanjeev Khanna. "New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS". In: *SODA*. SIAM, 2022, pp. 3529–3566. DOI: 10.1137/1.9781611977073.140 (cit. on p. 37).

[BKS23a] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Dynamic $(1+\varepsilon)$-Approximate Matching Size in Truly Sublinear Update Time". In: *FOCS*. IEEE, 2023, pp. 1563–1588 (cit. on p. 45).

[BKS23b] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Dynamic Algorithms for Packing-Covering LPs via Multiplicative Weight Updates". In: *SODA*. SIAM, 2023, pp. 1–47. DOI: 10.1137/1.9781611977554.CH1 (cit. on p. 37).

[BKS23c] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Sublinear Algorithms for $(1.5+\epsilon)$-Approximate Matching". In: *STOC*. ACM, 2023, pp. 254–266. DOI: 10.1145/3564246.3585252 (cit. on p. 38).

[BKSW22] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. "Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time". In: *CoRR* abs/2207.07438 (2022) (cit. on p. 45).

[BKSW23] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. "Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time". In: *SODA*. SIAM, 2023, pp. 100–128. DOI: 10.1137/1.9781611977554.CH5 (cit. on pp. 37, 38).

[BLLSSSW21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*. ACM, 2021, pp. 859–869. DOI: 10.1145/3406325.3451108 (cit. on pp. 13, 15).

[BLNPSSSW20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs". In: *FOCS*. IEEE, 2020, pp. 919–930. DOI: 10.1109/FOCS46700.2020.00090 (cit. on pp. 13, 27, 31).

[BMM12] Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. "Online Graph Edge-Coloring in the Random-Order Arrival Model." In: *Theory of Computing* 8.1 (2012), pp. 567–595 (cit. on p. 40).

[BMN92] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. "The greedy algorithm is optimal for on-line edge coloring". In: *Information Processing Letters (IPL)* 44.5 (1992), pp. 251–253 (cit. on pp. 12, 40).

[BN21] Gal Beniamini and Noam Nisan. "Bipartite perfect matching as a real polynomial". In: *STOC*. ACM, 2021, pp. 1118–1131 (cit. on pp. 34, 35).

[BPS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. "Deterministic Decremental Reachability, SCC, and Shortest Paths via Directed Expanders and Congestion Balancing". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 1123–1134. DOI: 10.1109/FOCS46700.2020.00108 (cit. on pp. 17, 37, 43).

[BRR23] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Sublinear Time Algorithms and Complexity of Approximate Maximum Matching". In: *STOC*. ACM, 2023, pp. 267–280. DOI: 10.1145/3564246.3585231 (cit. on pp. 37, 38).

[BS15] Aaron Bernstein and Cliff Stein. "Fully Dynamic Matching in Bipartite Graphs". In: *ICALP (1)*. Vol. 9134. Lecture Notes in Computer Science. Springer, 2015, pp. 167–179. DOI: 10.1007/978-3-662-47672-7_14 (cit. on pp. 38, 39).

[BS16] Aaron Bernstein and Cliff Stein. "Faster Fully Dynamic Matchings with Small Approximation Ratios". In: *SODA*. SIAM, 2016, pp. 692–711. DOI: 10.1137/1.9781611974331.CH50 (cit. on pp. 37, 38).

[BW09] Nikhil Bansal and Ryan Williams. "Regularity Lemmas and Combinatorial Algorithms". In: *FOCS*. IEEE Computer Society, 2009, pp. 745–754 (cit. on p. 14).

[CCK21] Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. "A Polynomial Lower Bound on the Number of Rounds for Parallel Submodular Function Minimization and Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2021, pp. 37–48. DOI: 10.1109/FOCS52979.2021.00013 (cit. on pp. 10, 24, 30, 44).

[CGHPS20] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. "Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 1135–1146. DOI: 10.1109/FOCS46700.2020.00109 (cit. on p. 13).

[CGJS22] Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. "Improved Lower Bounds for Submodular Function Minimization". In: *FOCS*. IEEE, 2022, pp. 245–254. DOI: 10.1109/FOCS54457.2022.00030 (cit. on p. 44).

[CGLNPS20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. "A Deterministic Algorithm for Balanced Cut with Applications to Dynamic Connectivity, Flows, and Beyond". In: *FOCS*. IEEE, 2020, pp. 1158–1167. DOI: 10.1109/FOCS46700.2020.00111 (cit. on p. 26).

[CHLP23] Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. "Steiner Connectivity Augmentation and Splitting-off in Poly-logarithmic Maximum Flows". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*. SIAM, 2023, pp. 2449–2488. DOI: 10.1137/1.9781611977554.CH95 (cit. on p. 13).

[Chr23] Aleksander Bjørn Grodt Christiansen. "The power of multi-step Vizing chains". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 1013–1026 (cit. on p. 40).

[CK24a] Julia Chuzhoy and Sanjeev Khanna. "A Faster Combinatorial Algorithm for Maximum Bipartite Matching". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*. SIAM, 2024, pp. 2185–2235 (cit. on pp. 10, 14–16, 32).

[CK24b] Julia Chuzhoy and Sanjeev Khanna. "Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm". In: *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2024*. ACM, 2024 (cit. on pp. 10, 14–16, 31, 32, 43).

[CKLPGS22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *FOCS*. IEEE, 2022, pp. 612–623. DOI: 10.1109/FOCS54457.2022.00064 (cit. on pp. 5, 14, 15, 26, 27, 32, 43).

[CKMST11] Paul F. Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. "Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs". In: *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*. ACM, 2011, pp. 273–282. DOI: 10.1145/1993636.1993674 (cit. on p. 13).

[CKPSSY21] Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. "Almost optimal super-constant-pass streaming lower bounds for reachability". In: *STOC*. ACM, 2021, pp. 570–583 (cit. on p. 31).

[CLNPSQ21] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. "Minimum Cuts in Directed Graphs via Partial Sparsification". In: *FOCS*. IEEE, 2021, pp. 1147–1158. DOI: 10.1109/FOCS52979.2021.00113 (cit. on p. 13).

[CLSSW19] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. "Faster Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2019, pp. 1146–1168. DOI: 10.1109/FOCS.2019.00072 (cit. on pp. 10, 23–27, 29, 30, 44).

[CPW19] Ilan Reuven Cohen, Binghui Peng, and David Wajc. "Tight Bounds for Online Edge Coloring". In: *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 1–25 (cit. on pp. 40, 41).

[CS18] Moses Charikar and Shay Solomon. "Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier". In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 33:1–33:14. DOI: 10.4230/LIPICS.ICALP.2018.33 (cit. on p. 37).

[Cun86] William H. Cunningham. "Improved Bounds for Matroid Partition and Intersection Algorithms". In: *SIAM J. Comput.* 15.4 (1986), pp. 948–957. DOI: 10.1137/0215066 (cit. on pp. 23, 27).

[DGGP19] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. "Fully dynamic spectral vertex sparsifiers and applications". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*. ACM, 2019, pp. 914–925. DOI: 10.1145/3313276.3316379 (cit. on p. 13).

[Din70] Efim A Dinic. "Algorithm for solution of a problem of maximum flow in networks with power estimation". In: *Soviet Math. Doklady*. Vol. 11. 1970, pp. 1277–1280 (cit. on pp. 14, 15, 27, 32).

[DNO19] Shahar Dobzinski, Noam Nisan, and Sigal Oren. "Economic efficiency requires interaction". In: *Games Econ. Behav.* 118 (2019), pp. 589–608 (cit. on pp. 11, 31, 34, 35).

[DS08] Samuel I. Daitch and Daniel A. Spielman. "Faster approximate lossy generalized flow via interior point algorithms". In: *STOC*. ACM, 2008, pp. 451–460 (cit. on p. 13).

[Edm70] Jack Edmonds. "Submodular functions, matroids, and certain polyhedra". In: *Combinatorial structures and their applications*. Gordon and Breach, 1970, pp. 69–87 (cit. on pp. 9, 27).

[Edm79] Jack Edmonds. "Matroid intersection". In: *Annals of discrete Mathematics*. Vol. 4. Elsevier, 1979, pp. 39–49 (cit. on p. 27).

[EDVJ68] Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. "Matroid partition". In: *50 Years of Integer Programming 1958–2008* (1968), p. 199 (cit. on pp. 9, 27).

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (1972), pp. 248–264. DOI: 10.1145/321694.321699 (cit. on pp. 5, 14, 15).

[ET75] Shimon Even and Robert Endre Tarjan. “Network Flow and Testing Graph Connectivity”. In: *SIAM J. Comput.* 4.4 (1975), pp. 507–518. DOI: 10.1137/0204043 (cit. on pp. 10, 15).

[FF56] Lester R. Ford and Delbert R. Fulkerson. “Maximal flow through a network”. In: *Canadian journal of Mathematics* 8.3 (1956), pp. 399–404 (cit. on pp. 5, 9, 14, 15).

[FGLPSY21] Sebastian Forster, Gramoz Goranci, Yang P. Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. “Minor Sparsifiers and the Distributed Laplacian Paradigm”. In: *FOCS*. IEEE, 2021, pp. 989–999 (cit. on p. 31).

[FGT21] Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. “Bipartite Perfect Matching is in Quasi-NC”. In: *SIAM J. Comput.* 50.3 (2021). DOI: 10.1137/16M1097870 (cit. on pp. 30, 31).

[Fin18] Jeremy T. Fineman. “Nearly work-efficient parallel algorithm for digraph reachability”. In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*. ACM, 2018, pp. 457–470 (cit. on p. 43).

[FKMSZ05] 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 (cit. on pp. 35, 45).

[Fre75] David A Freedman. “On Tail Probabilities for Martingales”. In: *The Annals of Probability* (1975) (cit. on p. 41).

[FS89] Greg N. Frederickson and Mandayam A. Srinivas. “Algorithms and Data Structures for an Expanded Family of Matroid Intersection Problems”. In: *SIAM J. Comput.* 18.1 (1989), pp. 112–138. DOI: 10.1137/0218008 (cit. on p. 25).

[Gab17] Harold N. Gabow. “The Weighted Matching Approach to Maximum Cardinality Matching”. In: *Fundam. Informaticae* 154.1-4 (2017), pp. 109–130. DOI: 10.3233/FI-2017-1555 (cit. on p. 44).

[Gab91] Harold N. Gabow. “A Matroid Approach to Finding Edge Connectivity and Packing Arborescences”. In: *STOC*. ACM, 1991, pp. 112–122. DOI: 10.1145/103418.103436 (cit. on pp. 6, 21, 24, 25).

[Gab95] Harold N. Gabow. “A Matroid Approach to Finding Edge Connectivity and Packing Arborescences”. In: *J. Comput. Syst. Sci.* 50.2 (1995), pp. 259–273. DOI: 10.1006/jcss.1995.1022 (cit. on pp. 21, 26).

[GG17] Shafi Goldwasser and Ofer Grossman. “Bipartite Perfect Matching in Pseudo-Deterministic NC”. In: *ICALP*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 87:1–87:13 (cit. on p. 31).

[GH61] Ralph E Gomory and Tien Chung Hu. “Multi-terminal network flows”. In: *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 551–570 (cit. on p. 13).

[GHKKTW15] Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Pushmeet Kohli, Robert Endre Tarjan, and Renato F. Werneck. "Faster and More Dynamic Maximum Flow by Incremental Breadth-First Search". In: *ESA*. Vol. 9294. Lecture Notes in Computer Science. Springer, 2015, pp. 619–630. DOI: 10.1007/978-3-662-48350-3_52 (cit. on p. 5).

[GKK12] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. "On the communication and streaming complexity of maximum bipartite matching". In: *SODA*. SIAM, 2012, pp. 468–485 (cit. on p. 35).

[GKKT15] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. "Dynamic graph connectivity with improved worst case update time and sublinear space". In: *CoRR* abs/1509.06464 (2015). arXiv: 1509.06464 (cit. on p. 26).

[GLP21] Yu Gao, Yang P. Liu, and Richard Peng. "Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao". In: *FOCS*. IEEE, 2021, pp. 516–527. DOI: 10.1109/FOCS52979.2021.00058 (cit. on pp. 13, 15).

[GLSSS19] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. "$(1 + \epsilon)$-Approximate Incremental Matching in Constant Deterministic Amortized Time". In: *SODA*. SIAM, 2019, pp. 1886–1898. DOI: 10.1137/1.9781611975482.114 (cit. on p. 38).

[GO16] Venkatesan Guruswami and Krzysztof Onak. "Superlinear Lower Bounds for Multipass Graph Processing". In: *Algorithmica* 76.3 (2016), pp. 654–683 (cit. on pp. 31, 35, 45).

[Gol08] Andrew V. Goldberg. "The Partial Augment-Relabel Algorithm for the Maximum Flow Problem". In: *ESA*. Vol. 5193. Lecture Notes in Computer Science. Springer, 2008, pp. 466–477 (cit. on pp. 14, 18).

[GP13] Manoj Gupta and Richard Peng. "Fully Dynamic (1+ e)-Approximate Matchings". In: *FOCS*. IEEE Computer Society, 2013, pp. 548–557. DOI: 10.1109/FOCS.2013.65 (cit. on pp. 37, 39, 45).

[GR98] Andrew V. Goldberg and Satish Rao. "Beyond the Flow Decomposition Barrier". In: *J. ACM* 45.5 (1998), pp. 783–797. DOI: 10.1145/290179.290181 (cit. on pp. 14, 15).

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*. SIAM, 2021, pp. 2212–2228. DOI: 10.1137/1.9781611976465.132 (cit. on pp. 10, 17).

[GS85] Harold N. Gabow and Matthias F. M. Stallmann. "Efficient Algorithms for Graphic Matroid Intersection and Parity (Extended Abstract)". In: *ICALP*. Vol. 194. Lecture Notes in Computer Science. Springer, 1985, pp. 210–220. DOI: 10.1007/BFb0015746 (cit. on pp. 6, 21, 25, 26).

[GSSU22] 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: *SOSA*. SIAM, 2022, pp. 12–23. DOI: 10.1137/1.9781611977066.2 (cit. on p. 38).

[GT20] Rohit Gurjar and Thomas Thierauf. "Linear Matroid Intersection is in Quasi-NC". In: *Comput. Complex.* 29.2 (2020), p. 9. DOI: 10.1007/s00037-020-00200-z (cit. on p. 30).

[GT79] Harold N. Gabow and Robert Endre Tarjan. "Efficient Algorithms for Simple Matroid Intersection Problems". In: *FOCS*. IEEE Computer Society, 1979, pp. 196–204. DOI: 10.1109/SFCS.1979.14 (cit. on p. 25).

[GT88] Andrew V. Goldberg and Robert Endre Tarjan. "A new approach to the maximum-flow problem". In: *J. ACM* 35.4 (1988), pp. 921–940. DOI: 10.1145/48014.61051 (cit. on pp. 5, 10, 14–16, 18).

[Gup14] Manoj Gupta. "Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic Update Time". In: *FSTTCS*. Vol. 29. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 227–239. DOI: 10.4230/LIPICS.FSTTCS.2014.227 (cit. on p. 37).

[GW88] Harold Gabow and Herbert Westermann. "Forests, frames, and games: algorithms for matroid sums and applications". In: *STOC*. 1988, pp. 407–421. DOI: 10.1145/62212.62252 (cit. on pp. 6, 21, 24–26).

[GX89] Harold N. Gabow and Ying Xu. "Efficient Algorithms for Independent Assignments on Graphic and Linear Matroids". In: *FOCS*. IEEE Computer Society, 1989, pp. 106–111. DOI: 10.1109/SFCS.1989.63463 (cit. on pp. 6, 21, 25, 26).

[Har08a] Nicholas J. A. Harvey. "Matroid intersection, pointer chasing, and Young's seminormal representation of S_n". In: *SODA*. SIAM, 2008, pp. 542–549 (cit. on p. 25).

[Har08b] Nicholas James Alexander Harvey. "Matchings, matroids and submodular functions". PhD thesis. Massachusetts Institute of Technology, 2008 (cit. on pp. 10, 27).

[Har09] Nicholas J. A. Harvey. "Algebraic Algorithms for Matching and Matroid Problems". In: *SIAM J. Comput.* 39.2 (2009), pp. 679–702. DOI: 10.1137/070684008 (cit. on p. 26).

[HHLRS24] Bernhard Haeupler, D. Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. "Low-Step Multi-commodity Flow Emulators". In: *STOC*. ACM, 2024, pp. 71–82. DOI: 10.1145/3618260.3649689 (cit. on p. 43).

[HK73] 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. DOI: 10.1137/0202019 (cit. on pp. 5, 10, 14, 16, 27, 31, 32, 35).

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture". In: *STOC*. ACM, 2015, pp. 21–30. DOI: 10.1145/2746539.2746609 (cit. on p. 37).

[HKPW23] Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. "Maintaining Expander Decompositions via Sparse Cuts". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*. SIAM, 2023, pp. 48–69. DOI: 10.1137/1.9781611977554.CH2 (cit. on p. 17).

[HMT88] András Hajnal, Wolfgang Maass, and György Turán. "On the Communication Complexity of Graph Properties". In: *STOC*. ACM, 1988, pp. 186–191. DOI: 10.1145/62212.62228 (cit. on pp. 11, 34, 35).

[HRVZ15] Zengfeng Huang, Bozidar Radunovic, Milan Vojnovic, and Qin Zhang. "Communication Complexity of Approximate Matching in Distributed Graphs". In: *STACS*. Vol. 30. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 460–473 (cit. on p. 35).

[HRVZ20] Zengfeng Huang, Bozidar Radunovic, Milan Vojnovic, and Qin Zhang. "Communication complexity of approximate maximum matching in the message-passing model". In: *Distributed Comput.* 33.6 (2020), pp. 515–531 (cit. on p. 35).

[IKLSW12] Gábor Ivanyos, Hartmut Klauck, Troy Lee, Miklos Santha, and Ronald de Wolf. "New bounds on the classical and quantum communication complexity of some graph properties". In: *FSTTCS*. Vol. 18. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 148–159 (cit. on pp. 11, 31, 34, 35).

[IM81] Oscar H Ibarra and Shlomo Moran. "Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication". In: *Information Processing Letters* 13.1 (1981), pp. 12–15 (cit. on p. 32).

[JB65] CGJ Jacobi and Carl Wilhelm Borchardt. "De investigando ordine systematis aequationum differentialium vulgarium cujuscunque." In: (1865) (cit. on p. 9).

[JJST22] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. "Regularized Box-Simplex Games and Dynamic Decremental Bipartite Matching". In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 77:1–77:20. DOI: 10.4230/LIPICS.ICALP.2022.77 (cit. on p. 37).

[JST20] Yujia Jin, Aaron Sidford, and Kevin Tian. "Semi-Streaming Bipartite Matching in Fewer Passes and Less Space". In: *CoRR* abs/2011.03495 (2020) (cit. on p. 31).

[Kah96] Jeff Kahn. "Asymptotically good list-colorings". In: *Journal of Combinatorial Theory, Series A* 73.1 (1996), pp. 1–59 (cit. on p. 40).

[Kap21] Michael Kapralov. “Space Lower Bounds for Approximating Maximum Matching in the Edge Arrival Model”. In: *SODA*. SIAM, 2021, pp. 1874–1893 (cit. on p. 35).

[Kar73] Alexander V Karzanov. “On finding maximum flows in networks with special structure and some applications”. In: *Matematicheskie Voprosy Upravleniya Proizvodstvom* 5 (1973), pp. 81–94 (cit. on pp. 10, 14, 15).

[Kis22] Peter Kiss. “Deterministic Dynamic Matching in Worst-Case Update Time”. In: *ITCS*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 94:1–94:21. DOI: 10.4230/LIPICS.ITCS.2022.94 (cit. on p. 38).

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. “Dynamic graph connectivity in polylogarithmic worst case time”. In: *SODA*. SIAM, 2013, pp. 1131–1142. DOI: 10.1137/1.9781611973105.81 (cit. on p. 26).

[KKS14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. “Approximating matching size from random streams”. In: *SODA*. SIAM, 2014, pp. 734–751 (cit. on p. 35).

[KL15] David R. Karger and Matthew S. Levine. “Fast Augmenting Paths by Random Sampling from Residual Graphs”. In: *SIAM J. Comput.* 44.2 (2015), pp. 320–339. DOI: 10.1137/070705994 (cit. on p. 14).

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. “An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations”. In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*. SIAM, 2014, pp. 217–226. DOI: 10.1137/1.9781611973402.16 (cit. on pp. 13, 16).

[KLS20] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. “Unit Capacity Maxflow in Almost $O(m^{4/3})$ Time”. In: *FOCS*. IEEE, 2020, pp. 119–130. DOI: 10.1109/FOCS46700.2020.00020 (cit. on pp. 13, 15).

[KLSST22] Janardhan Kulkarni, Yang P Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. “Online Edge Coloring via Tree Recurrences and Correlation Decay”. In: *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*. 2022, pp. 2958–2977 (cit. on pp. 40, 41).

[KMNT20] Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. “Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples”. In: *SODA*. SIAM, 2020, pp. 1753–1772 (cit. on p. 35).

[KMP12] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. “Faster approximate multicommodity flow using quadratically coupled flows”. In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*. ACM, 2012, pp. 1–18. DOI: 10.1145/2213977.2213979 (cit. on p. 13).

[KMT21] Michael Kapralov, Gilbert Maystre, and Jakab Tardos. "Communication Efficient Coresets for Maximum Matching". In: *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*. SIAM, 2021, pp. 156–164 (cit. on p. 35).

[KP22] Shimon Kogan and Merav Parter. "New Diameter-Reducing Shortcuts and Directed Hopsets: Breaking the Barrier". In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*. SIAM, 2022, pp. 1326–1341. DOI: 10.1137/1.9781611977073.55 (cit. on p. 43).

[Kuh55] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97 (cit. on p. 9).

[KUW85] Richard M. Karp, Eli Upfal, and Avi Wigderson. "Constructing a Perfect Matching is in Random NC". In: *STOC*. ACM, 1985, pp. 22–32 (cit. on pp. 30, 31, 44).

[KVKV11] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Vol. 1. Springer, 2011 (cit. on p. 36).

[KVV90] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. "An Optimal Algorithm for On-line Bipartite Matching". In: *STOC*. ACM, 1990, pp. 352–358 (cit. on p. 31).

[Law75] Eugene L. Lawler. "Matroid intersection algorithms". In: *Math. Program.* 9.1 (1975), pp. 31–56. DOI: 10.1007/BF01681329 (cit. on p. 27).

[LJS19] Yang P. Liu, Arun Jambulapati, and Aaron Sidford. "Parallel Reachability in Almost Linear Work and Square Root Depth". In: *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*. IEEE Computer Society, 2019, pp. 1664–1686 (cit. on p. 43).

[LL15] Cedric Yen-Yu Lin and Han-Hsuan Lin. "Upper Bounds on Quantum Query Complexity Inspired by the Elitzur-Vaidman Bomb Tester". In: *Computational Complexity Conference*. Vol. 33. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 537–566 (cit. on pp. 34, 35).

[LNPSY21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. "Vertex connectivity in poly-logarithmic max-flows". In: *STOC*. ACM, 2021, pp. 317–329. DOI: 10.1145/3406325.3451088 (cit. on p. 13).

[Lou10] Anand Louis. "Cut-Matching Games on Directed Graphs". In: *CoRR* abs/1010.1047 (2010). arXiv: 1010.1047 (cit. on p. 18).

[Lov79] László Lovász. "On determinants, matchings, and random algorithms". In: *FCT*. Akademie-Verlag, Berlin, 1979, pp. 565–574 (cit. on pp. 29, 31).

[LP20] Jason Li and Debmalya Panigrahi. "Deterministic Min-cut in Poly-logarithmic Max-flows". In: *FOCS*. IEEE, 2020, pp. 85–92. DOI: 10.1109/FOCS46700.2020.00017 (cit. on p. 13).

[LS14] Yin Tat Lee and Aaron Sidford. "Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}$(sqrt(rank)) Iterations and Faster Algorithms for Maximum Flow". In: *FOCS*. 2014, pp. 424–433. DOI: 10.1109/FOCS.2014.52 (cit. on pp. 13, 15, 27).

[LS20] Yang P. Liu and Aaron Sidford. "Faster energy maximization for faster maximum flow". In: *STOC*. ACM, 2020, pp. 803–814 (cit. on p. 13).

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. "A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization". In: *FOCS*. IEEE Computer Society, 2015, pp. 1049–1065. DOI: 10.1109/FOCS.2015.68 (cit. on pp. 10, 23, 27).

[Mad13] Aleksander Madry. "Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back". In: *FOCS*. IEEE Computer Society, 2013, pp. 253–262. DOI: 10.1109/FOCS.2013.35 (cit. on pp. 13, 31, 32).

[Mad16] Aleksander Madry. "Computing maximum flow with augmenting electrical flows". In: *FOCS*. IEEE. 2016, pp. 593–602. DOI: 10.1109/FOCS.2016.70 (cit. on pp. 13, 31).

[MS04] Marcin Mucha and Piotr Sankowski. "Maximum Matchings via Gaussian Elimination". In: *FOCS*. IEEE Computer Society, 2004, pp. 248–255 (cit. on p. 31).

[Mun57] James Munkres. "Algorithms for the assignment and transportation problems". In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38 (cit. on p. 9).

[Ngu19] Huy L. Nguyen. "A note on Cunningham's algorithm for matroid intersection". In: *CoRR* abs/1904.04129 (2019) (cit. on pp. 23, 27).

[Nis21] Noam Nisan. "The Demand Query Model for Bipartite Matching". In: *SODA*. SIAM, 2021, pp. 592–599 (cit. on pp. 11, 31, 34, 35).

[NS16] Ofer Neiman and Shay Solomon. "Simple Deterministic Algorithms for Fully Dynamic Maximal Matching". In: *ACM Trans. Algorithms* 12.1 (2016). Announced at STOC'13, 7:1–7:15. DOI: 10.1145/2700206 (cit. on p. 37).

[NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. "Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time". In: *FOCS*. IEEE Computer Society, 2017, pp. 950–961. DOI: 10.1109/FOCS.2017.92 (cit. on p. 26).

[OR10] Krzysztof Onak and Ronitt Rubinfeld. "Maintaining a large matching and a small vertex cover". In: *STOC*. ACM, 2010, pp. 457–464. DOI: 10.1145/1806689.1806753 (cit. on p. 37).

[Pen16] Richard Peng. "Approximate Undirected Maximum Flows in $O(m\text{polylog}(n))$ Time". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*. SIAM, 2016, pp. 1862–1867. DOI: 10.1137/1.9781611974331.ch130 (cit. on pp. 13, 16).

[PF24] Laurent Perron and Vincent Furnon. *OR-Tools*. Version v9.11. Google, May 7, 2024 (cit. on p. 5).

[PS16] David Peleg and Shay Solomon. "Dynamic $(1 + \epsilon)$-Approximate Matchings: A Density-Sensitive Approach". In: *SODA*. SIAM, 2016, pp. 712–729. DOI: 10.1137/1.9781611974331.CH51 (cit. on p. 37).

[PT07] Mihai Puatracscu and Mikkel Thorup. "Planning for Fast Connectivity Updates". In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*. IEEE Computer Society, 2007, pp. 263–271. DOI: 10.1109/FOCS.2007.54 (cit. on pp. 10, 17).

[Qua23] Kent Quanrud. "Faster exact and approximation algorithms for packing and covering matroids via push-relabel". In: *CoRR* abs/2303.01478 (2023). DOI: 10.48550/arXiv.2303.01478. arXiv: 2303.01478 (cit. on pp. 11, 24, 27).

[Qua24] Kent Quanrud. "Adaptive Sparsification for Matroid Intersection". In: *ICALP*. Vol. 297. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 118:1–118:20. DOI: 10.4230/LIPICS.ICALP.2024.118 (cit. on p. 27).

[Räc02] Harald Räcke. "Minimizing Congestion in General Networks". In: *43rd Symposium on Foundations of Computer Science (FOCS 2002)*. IEEE Computer Society, 2002, pp. 43–52. DOI: 10.1109/SFCS.2002.1181881 (cit. on pp. 10, 17).

[Raz92] Alexander A. Razborov. "On the Distributional Complexity of Disjointness". In: *Theor. Comput. Sci.* 106.2 (1992), pp. 385–390 (cit. on p. 35).

[Rot82] Alvin E. Roth. "The Economics of Matching: Stability and Incentives". In: *Math. Oper. Res.* 7.4 (1982), pp. 617–628 (cit. on p. 35).

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. "Computing Cut-Based Hierarchical Decompositions in Almost Linear Time". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*. SIAM, 2014, pp. 227–238. DOI: 10.1137/1.9781611973402.17 (cit. on pp. 10, 17).

[RSW22] Mohammad Roghani, Amin Saberi, and David Wajc. "Beating the Folklore Algorithm for Dynamic Matching". In: *ITCS*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 111:1–111:23 (cit. on p. 31).

[RT85] James Roskind and Robert E. Tarjan. "A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees". In: *Math. Oper. Res.* 10.4 (1985), pp. 701–708. DOI: 10.1287/moor.10.4.701 (cit. on p. 25).

[San07] Piotr Sankowski. "Faster dynamic matchings and vertex connectivity". In: *SODA*. SIAM, 2007, pp. 118–126 (cit. on p. 37).

[She13] Jonah Sherman. "Nearly Maximum Flows in Nearly Linear Time". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*. IEEE Computer Society, 2013, pp. 263–269. DOI: 10.1109/FOCS.2013.36 (cit. on pp. 13, 16).

[She17] Jonah Sherman. "Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*. ACM, 2017, pp. 452–460. DOI: 10.1145/3055399.3055501 (cit. on pp. 13, 16).

[Sol16] Shay Solomon. "Fully Dynamic Maximal Matching in Constant Update Time". In: *FOCS*. IEEE Computer Society, 2016, pp. 325–334. DOI: 10.1109/FOCS.2016.43 (cit. on p. 37).

[SP24] Aurelio L. Sulser and Maximilian Probst Gutenberg. "A Simple and Near-Optimal Algorithm for Directed Expander Decompositions". In: *CoRR* abs/2403.04542 (2024). DOI: 10.48550/ARXIV.2403.04542. arXiv: 2403.04542 (cit. on p. 17).

[ST04] Daniel A. Spielman and Shang-Hua Teng. "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. ACM, 2004, pp. 81–90. DOI: 10.1145/1007352.1007372 (cit. on p. 13).

[ST18] Aaron Sidford and Kevin Tian. "Coordinate Methods for Accelerating ℓ_∞ Regression and Faster Approximate Maximum Flow". In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*. IEEE Computer Society, 2018, pp. 922–933. DOI: 10.1109/FOCS.2018.00091 (cit. on pp. 13, 16).

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. "A Data Structure for Dynamic Trees". In: *J. Comput. Syst. Sci.* 26.3 (1983), pp. 362–391. DOI: 10.1016/0022-0000(83)90006-5 (cit. on p. 18).

[SW21] Amin Saberi and David Wajc. "The Greedy Algorithm is *not* Optimal for On-Line Edge Coloring". In: *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP)*. 2021, 109:1–109:18 (cit. on p. 40).

[Ten02] Moshe Tennenholtz. "Tractable combinatorial auctions and b-matching". In: *Artif. Intell.* 140.1/2 (2002), pp. 231–243 (cit. on p. 35).

[Vai89] Pravin M. Vaidya. "A New Algorithm for Minimizing Convex Functions over Convex Sets (Extended Abstract)". In: *FOCS*. IEEE Computer Society, 1989, pp. 338–343 (cit. on p. 36).

[Viz64] Vadim G Vizing. "On an estimate of the chromatic class of a p-graph". In: *Diskret analiz* 3 (1964), pp. 25–30 (cit. on p. 39).

[Waj20] David Wajc. "Rounding dynamic matchings against an adaptive adversary". In: *STOC*. ACM, 2020, pp. 194–207. DOI: 10.1145/3357713.3384258 (cit. on pp. 31, 37).

[Wel76] Dominic JA Welsh. *Matroid theory*. Academic Press, London, New York, 1976 (cit. on pp. 10, 25).

[XG94] Ying Xu and Harold N. Gabow. "Fast Algorithms for Transversal Matroid Intersection Problems". In: *ISAAC*. Vol. 834. Lecture Notes in Computer Science. Springer, 1994, pp. 625–633. DOI: 10.1007/3-540-58325-4_231 (cit. on pp. 21, 25, 26).

[Yos] Yosupo. *Library Checker*. https://judge.yosupo.jp/. Accessed: Sep 2024 (cit. on p. 5).

[Zha04] Shengyu Zhang. "On the Power of Ambainis's Lower Bounds". In: *ICALP*. Vol. 3142. Lecture Notes in Computer Science. Springer, 2004, pp. 1238–1250 (cit. on pp. 31, 34, 35).

Part II

Included Papers

Paper A

Maximum Flow by Augmenting Paths in $n^{2+o(1)}$ Time

Aaron Bernstein, Joakim Blikstad,
Thatchaphol Saranurak, Ta-Wei Tu

Article published in 65rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27—37, 2024. [BBST24]
Full version at `https://arxiv.org/abs/2406.03648`.

Abstract

We present a combinatorial algorithm for computing exact maximum flows in directed graphs with n vertices and edge capacities from $\{1, \ldots, U\}$ in $n^{2+o(1)} \log U$ time, which is almost optimal in dense graphs. Our algorithm is a novel implementation of the classical augmenting-path framework; we list augmenting paths more efficiently using a new variant of the push-relabel algorithm that uses additional edge weights to guide the algorithm, and we derive the edge weights by constructing a directed expander hierarchy.

Even in unit-capacity graphs, this breaks the long-standing $O(m\sqrt{m})$ and $O(mn^{2/3})$ time bounds of the previous combinatorial algorithms by Karzanov (1973) and Even and Tarjan (1975) when the graph has $m = \omega(n^{4/3})$ edges. Notably, our approach does not rely on continuous optimization nor heavy dynamic graph data structures, both of which are crucial in the recent developments that led to the almost-linear time algorithm by Chen et al. (FOCS 2022). Our running time also matches the $n^{2+o(1)}$ time bound of the independent combinatorial algorithm by Chuzhoy and Khanna (STOC 2024) for computing the maximum bipartite matching, a special case of maximum flow.

A.1 Introduction

Fast algorithms for computing maximum flows have played a central role in algorithmic research, motivating various algorithmic paradigms such as graph sparsification, dynamic data structures, and the use of continuous optimization in combinatorial problems. These algorithms also have numerous applications in problems like bipartite matching, minimum cuts, and Gomory-Hu trees [GH61; LP20; LNPSY21; CLNPSQ21; CHLP23; ALPS23]. Below, we summarize the development of fast maximum flow algorithms following Dantzig's introduction of the problem [Dan51]. The input graph for this problem is a directed graph with n vertices and m edges. For convenience, only in this introduction, we assume that edge capacities range from $\{1, \ldots, \text{poly}(n)\}$.

Augmenting Paths. Ford and Fulkerson [FF56] first introduced the *augmenting path* framework. In this framework, algorithms repeatedly find an augmenting path (or a collection of them) in the residual graph. They then augment the flow along this path with a value equal to the bottleneck of the augmenting path. Over the next four decades, this simple framework led to the development of several influential techniques, including shortest augmenting paths [EK72], blocking flows [Kar73; Din70; GR98], push-relabel [GT88], and sparsification [KL15]. The best time bound within this framework, $O(m \cdot \min\{m^{1/2}, n^{2/3}\})$, was given by Karzanov [Kar73] and independently by Even and Tarjan [ET75] for unit-capacity graphs. Goldberg and Rao [GR98] later matched this time bound in capacitated graphs up to poly-logarithmic factors.

Continuous Optimization and Dynamic Graph Data Structures. In the 2000s, Spielman and Teng [ST04] introduced a completely different framework based on *continuous optimization*, and solved the electrical flow problem in near-linear time.[1] Then, Daitch and Spielman [DS08] showed a reduction from maximum flow to electrical flow using an interior point method. This motivated further research on advanced interior point methods [Mad13; LS14; Mad16; LS20; KLS20] that minimize the number of iterations of calling electrical flow or related problems. As a result, a $\widetilde{O}(m\sqrt{n})$-time[2] maximum flow algorithm [LS14] and, for unit-capacity graphs, a $\widetilde{O}(m^{4/3})$-time algorithm [KLS20] were developed.

Since 2020, the focus has shifted from minimizing the number of iterations in interior point methods to instead minimizing the cost per iteration using *dynamic graph data structures*. Building upon dynamic data structures for sparsifiers [DGGP19; BBPNSSS22; CGHPS20], a series of impressive works [BLNPSSSW20; BLLSSSW21;

[1] Algorithms for *approximating* maximum flow in undirected graphs using gradient descent and multiplicative weight update also follow this framework [CKMST11; KMP12; She13; KLOS14; Pen16; She17; ST18].

[2] In this paper, we use $\widetilde{O}(\cdot)$ to hide poly-logarithmic factors in n and $\widehat{O}(\cdot)$ for subpolynomial factors.

GLP21; BGJLLPS22] finally led to the breakthrough by Chen *et al.* [CKLPGS22; BCPKLPSS23] who showed an $m^{1+o(1)}$-time algorithm for maximum flow and its generalization.

Combinatorial Approaches. Although the recent developments have achieved an almost optimal time bound, these algorithms are not simple in either conceptual or technical sense. Continuous optimization approaches update the flow solutions without a clear combinatorial interpretation, and the required dynamic data structures are still highly involved. With this motivation, Chuzhoy and Khanna recently [CK24a] showed a conceptually simpler algorithm for maximum bipartite matching, a special case of maximum flow, that runs in $\widetilde{O}(m^{1/3}n^{5/3})$ time; in very recent independent work, they improved the running time to $n^{2+o(1)}$ [CK24b].[3]

These algorithms update the flow (i.e., the fractional matching) in a more intuitive manner: they repeatedly increase the flow value along paths listed by dynamic shortest-path data structures. Moreover, the flow on each edge is a multiple of $\frac{1}{\Theta(\log n)}$, i.e., it is almost integral. Unfortunately, the algorithms do not extend to exact maximum flow, even in unit-capacity graphs.

So, can one hope for an optimal and combinatorial maximum flow algorithm? We make significant progress in this direction by showing that the classical augmenting path framework can provide an almost-optimal algorithm for dense graphs.

Theorem A.1.1. *There is an augmenting-path-based randomized algorithm that, given a directed graph with n vertices and edge capacities from $\{1, \ldots, U\}$, with high probability computes a maximum s-t flow in $n^{2+o(1)} \log U$ time.*

Our algorithm strictly follows the augmenting path framework, i.e., it maintains an *integral* flow that is repeatedly increased along augmenting paths in the residual graph. When $m = \omega(n^{4/3})$, Theorem A.1.1 improves upon the long-standing time bound of $O(m \cdot \min\{m^{1/2}, n^{2/3}\})$ presented in [Kar73; ET75; GR98], in the context of previous augmenting-path-based algorithms. Additionally, our running time matches the $n^{2+o(1)}$-time bound in the independent work by Chuzhoy and Khanna [CK24b] who gave combinatorial algorithms for computing the maximum bipartite matching (and consequently unit-vertex-capacitated maximum flow), a special case of maximum flow. As far as we know, their techniques are based on dynamic shortest-path data structures and multiplicative weights update and, hence, are different from ours.

A Bird's-Eye View of Our Algorithm. We now give a basic outline of our algorithm; we provide a more detailed overview in Section A.2. To list augmenting paths, we introduce the *weighted push-relabel* algorithm, a new and simple variant of the well-known push-relabel algorithm [GT88] guided by an additional edge

[3]The algorithm of [CK24b] was submitted several months before ours (STOC 2024), but we consider it independent because it was not publicly available when we submitted our paper to FOCS 2024.

weight function. Edges with higher weight are relabeled less often, allowing for more efficiency. Given a "good" weight function, the weighted push-relabel algorithm will list augmenting paths in $\widetilde{O}(n^2)$ total time and return an $O(1)$-approximate maximum flow. By repeating the algorithm on the residual graph, this immediately gives a maximum flow algorithm.

Our starting observation is that on a DAG with a topological order $\boldsymbol{\tau}$, the simple function $\boldsymbol{w}(u,v) = |\boldsymbol{\tau}_u - \boldsymbol{\tau}_v|$ for each edge (u,v) is in fact a good weight function. The question is now to figure out what "good" weight function to use on general graphs.

We introduce the *directed expander hierarchy* and show that it induces a natural vertex ordering $\boldsymbol{\tau}$ such that $\boldsymbol{w}(u,v) = |\boldsymbol{\tau}_u - \boldsymbol{\tau}_v|$ is a good weight function. We remark that while there are several successful variants of expander hierarchies in undirected graphs [Räc02; PT07; RST14; GRST21], we believe ours is the first paper to successfully apply them to directed graphs. Unfortunately, all known approaches for the hierarchy construction are either too slow [PT07], assume a maximum flow subroutine itself [RST14], or are specific to undirected graphs [GRST21].

Therefore, we show a new bottom-up construction based on our weighted push-relabel algorithm. The basic idea is that we repeatedly use weighted push-relabel to construct the next level of the hierarchy, which in turn gives us a better weight function, allowing us to compute yet one more level. To be a bit more concrete, let X_i be the candidate edge set for level-i of the hierarchy and suppose that we have already built a directed expander hierarchy of $G \setminus X_i$, which consists of all edges below level i. If we can certify that X_i is expanding (i.e. well-connected in some sense), then we can leave X_i as the last level of the hierarchy; on the other hand, if X_i if not expanding, then we need to find a sparse cut with respect to X_i and elevate those cut edges to the next level X_{i+1}. As is standard, we solve this problem using the cut-matching game, which requires computing flow between subsets of X_i. The challenge lies in solving this flow problem efficiently.

The crucial observation is that by setting the weight of edges in $G \setminus X_i$ according to its expander hierarchy (which we already computed) and setting the weight of all edges in X_i to be n, our weighted push-relabel algorithm will solve this flow problem in $\widetilde{O}(n^2)$ time. We note that standard push-relabel (without weights) can only solve this problem for the bottom level of the hierarchy, i.e., when X_i is the whole edge set.

We emphasize that so far, all of our algorithmic components (from Section A.4 to A.6) are very implementable. The weighted push-relabel algorithm (Algorithm A.1) simply increments vertex labels and, in capacitated graphs, uses the link-cut tree [ST83] to push flow. To find sparse cuts (Algorithm A.2), we also call Dijkstra's algorithm to produce levels and return the sparsest level cut.

The novelty is in the analysis. To show that the directed expander hierarchy gives a good weight function, we prove a new trade-off between length and congestion for rerouting flow on expanders. To show that there exists a sparse level cut, we show a novel angle to the *directed expander pruning* problem studied in [BPS20; HKPW23; SP24]. In the standard version of pruning, we update a few edges in

a directed ϕ-expander G, and the goal is to prune away a small set of vertices P so that $G \setminus P$ is still an expander. We extend pruning to work with *path-reversal* updates, which reverse the direction of *every* edge on a given path; this kind of update is very natural in the residual graph (reversing augmenting paths). We show that, somewhat surprisingly, reversing a whole path has approximately the same impact on pruning as updating a single edge. This allows us to show that the directed expander hierarchy is robust under flow augmentation (Lemma A.6.5). For our purposes, we only ever need an existential version of path-reversal pruning, but the algorithmic version should be plausible and useful.

There is unfortunately one challenge that adds a huge amount of complexity. When computing a sparse cut with respect to X_i, the sparse cut can "cut through" components of the expander hierarchy in the lower levels. This forbids us from building the hierarchy bottom-up in one go; instead, our algorithm needs to regularly move up edges at different levels of the hierarchy. To modularize the analysis, we employ a data structure point-of-view that models these interactions between levels. We note that our approach is not inherently *dynamic*, in that we are not aiming for fast or sublinear update times. In fact, each operation of the data structure requires $n^{2+o(1)}$ time. The usefulness of this perspective lies instead in showing that only $n^{o(1)}$ sequential updates are needed. This is by far the most complicated part of our algorithm (essentially all of Section A.7), and is also the only reason our algorithm is randomized and requires an inherent $n^{o(1)}$-factor in the running time. We believe this step can be simplified once tools related to directed expanders are as developed as their undirected counterparts [RST14; SW19; GRST21].

To summarize, in contrast to recent developments, Theorem A.1.1 does not rely on continuous optimization or heavy dynamic data structures. It also paves the way to a very implementable $\widetilde{O}(n^2)$-time deterministic algorithm once a better construction of directed expander hierarchy is shown. We note that our paper is quite self-contained: the black boxes we assume only include basic graph algorithms (e.g., topological sort and Dijkstra's algorithm), link-cut trees [ST83], and Louis's cut-matching game [Lou10]. Lastly, we believe and hope that some novel tools we developed in this paper—including the *weighted push-relabel algorithm*, *directed expander hierarchy*, and *expander pruning under path-reversals*—will find future applications.

Future Work. The natural next step is to show a simple algorithm for constructing a directed expander hierarchy in $\widetilde{O}(n^2)$. Combined with our new approach, this would yield a much simpler $\widetilde{O}(n^2)$-time max-flow algorithm, and we believe it would also prove a powerful tool for other directed problems.

A more challenging goal is to achieve an $m^{1+o(1)}$ time bound via simple combinatorial algorithms. One takeaway of our paper is that the main bottleneck seems to be a fast $n^{o(1)}$-approximation for DAGs. On the one hand, it seems quite plausible that our tools would allow an improvement for DAGs to be generalized to all directed graphs; but on the other hand, the current toolkit for DAGs is quite limited. DAGs

also seem to capture the hardness of other fundamental problems in directed graphs, such as dynamic shortest paths [BPS20], parallel reachability [Fin18; LJS19], and diameter-reducing shortcuts [KP22].

Organization. The rest of the paper is organized as follows. In Section A.2, we give a comprehensive overview of the technical components of our algorithm. In Section A.3 we provide necessary preliminaries. We develop our weighted push-relabel algorithm in Section A.4. In Section A.5, we show that a directed expander hierarchy induces a "good" weight function, and hence, when combined with our weighted push-relabel algorithm, solves maximum flow. In Section A.6, we show how to leverage the weighted push-relabel algorithm to compute sparse cuts, which is a crucial subroutine in how we construct the expander hierarchy in Section A.7. In Sections A.8 to A.10 we provide details omitted from the main body of the paper.

A.2 Technical Overview

In this section we give a high-level overview our maximum flow algorithm. For simplicity of presentation, we assume during this overview that the input graph is unit-capacitated. Note that it suffices to design a constant- or even $1/n^{o(1)}$-approximate flow algorithm for directed graphs, as the exact algorithm then follows by repeating the approximate algorithm $n^{o(1)}$ times on the residual graph. This is in contrast to undirected graphs: although efficient approximations are known here [She13; KLOS14; Pen16; She17; ST18], the residual graph of the found approximate flow is no longer undirected, so an approximate flow algorithm cannot be bootstrapped to an exact one. Although we assume unit capacities, in the analysis we will sometimes refer to a flow $\boldsymbol{f}$ that disobeys these capacities; we define the *congestion* of a flow $\boldsymbol{f}$, denoted $\text{cong}(\boldsymbol{f})$, to be $\max_{e\in E} \boldsymbol{f}(e)$.

A.2.1 Weighted Push-Relabel Algorithm

The starting point of our algorithm is a weighted variant of the classic push-relabel algorithm.

Summary of Classic Push-Relabel. Let us recall the classic push-relabel algorithm in unit-capacitated graphs. Suppose we have a flow instance with integral source vector $\boldsymbol{\Delta}$ and sink vector $\boldsymbol{\nabla}$, and let us assume for simplicity that this flow instance is feasible. The push-relabel algorithm will always maintain a pre-flow,[4] where every vertex v might have excess flow $\mathbf{ex}_{\boldsymbol{f}}(v) \coloneqq \max\{\boldsymbol{\Delta}(v) - \boldsymbol{f}^{\text{out}}(v) - \boldsymbol{\nabla}(v), 0\}$ where $\boldsymbol{f}^{\text{out}}(v)$ denotes the *net* flow going out from v; note that initially, all positive excess are on the source vertices. The algorithm also maintains an integral label $\boldsymbol{\ell}(v)$ on every $v \in V$, which gradually increases over time; initially $\boldsymbol{\ell}(v) = 0$ for all $v \in V$.

[4] A *pre-flow* is an intermediate flow that has not yet sent all units of demands to sink vertices.

Informally, the main loop of the push-relabel algorithm repeatedly finds a vertex v with $\mathbf{ex}_{\boldsymbol{f}}(v) > 0$ and attempts to *push* a unit of flow along an edge (v, w) with $\boldsymbol{\ell}(v) \geq \boldsymbol{\ell}(w) + 1$; we refer to such edges as *admissible*. Following the standard operation of residual graphs, this edge (v, w) is then removed and replaced with the reverse edge (w, v). If a vertex v has $\mathbf{ex}_{\boldsymbol{f}}(v) > 0$, but there are no admissible edges (v, w), then the algorithm performs operation RELABEL(v), which increases $\boldsymbol{\ell}(v)$ by 1. The sequence of push operations effectively traces augmenting paths from the source vertices to the sink vertices.

Analysis of Classic Push-Relabel. The analysis rests on the following admissibility invariant: for any edge (u, v) in the residual graph, we have $\boldsymbol{\ell}(u) \leq \boldsymbol{\ell}(v) + 1$. This easily follows from the fact that if $\boldsymbol{\ell}(u) = \boldsymbol{\ell}(v) + 1$, then (u, v) is an admissible edge, so the algorithm will not relabel u as long as (u, v) remains in the residual graph.

We now sketch the proof that push-relabel successfully finds a flow that routes all the demands. In fact, we show something stronger: at termination, we have $\boldsymbol{\ell}(v) \leq n$ for all $v \in V$. Say, for contradiction, that the algorithm relabels a vertex v from $\boldsymbol{\ell}(v) = n$ to $\boldsymbol{\ell}(v) = n + 1$. This implies that $\mathbf{ex}_{\boldsymbol{f}}(v) > 0$, so since we assumed the original flow instance is feasible, there must exist some path in the residual graph from v to an unsaturated sink vertex t. It is easy to see that $\boldsymbol{\ell}(t) = 0$; since t is still a sink, it never had an excess, and so was never relabeled. This (v, t)-path has at most $n-1$ edges, so by the admissibility invariant above, $\boldsymbol{\ell}(v) \leq \boldsymbol{\ell}(t) + n - 1 = n - 1$, contradicting the assumption that $\boldsymbol{\ell}(v) = n$.

For the running time analysis, it is not hard to check that any edge (u, v) can undergo at most one push operation as long as the level $\boldsymbol{\ell}(u)$ is fixed; similarly, the admissibility status of (u, v) can only change when u or v is relabeled. Since u and v undergo at most $O(n)$ relabel operations, the total running time is $O(mn)$.

Motivating Our Weighted Push-Relabel. Consider the following simplified scenario: we are told in advance that a certain subset of the edges is *infrequent*, meaning that there exists some approximate maximum flow $\boldsymbol{f}$, where every flow path in $\boldsymbol{f}$ uses at most k infrequent edges (think of k as small).

We then modify the classic push-relabel algorithm as follows. An infrequent edge (u, v) only counts as admissible if $\boldsymbol{\ell}(u) \geq \boldsymbol{\ell}(v) + n/k$. This means that the algorithm might need to perform more relabel operations, and yet we can still show that if any label ever exceeds $10n$, this means that push-relabel has already routed a constant fraction of the demand. To see this, assume, for contradiction, that the algorithm relabels some v from $\boldsymbol{\ell}(v) = 10n$ to $\boldsymbol{\ell}(v) = 10n + 1$. As before, there must exist some (v, t)-path P in the residual graph to an unsaturated sink t with $\boldsymbol{\ell}(t) = 0$. Intuitively, the path P contains at most k infrequent edges: this is not technically true, and the full analysis is slightly more involved. However, this is close to being true, so we make the simplifying assumption here that this path contains at most

$9k$ infrequent edges.[5] The natural generalization of the admissibility invariant then implies that

$$\begin{aligned}\boldsymbol{\ell}(v) &\leq \boldsymbol{\ell}(t) + [\#\text{ infrequent edges on } P] \cdot (n/k) + [\#\text{ frequent edges on } P] \\ &\leq 0 + 9k \cdot (n/k) + n - 1 = 10n - 1,\end{aligned}$$

contradicting the assumption that $\boldsymbol{\ell}(v) = 10n$. For the running time analysis, note that an infrequent edge can only change status every n/k relabels, so the new runtime is $O([\#\text{ infrequent edges}] \cdot n/k + [\#\text{ frequent edges}] \cdot n)$, which is significantly smaller than $O(mn)$ if most edges are infrequent.

Weighted Push-Relabel. Imagine a generalization of the above scenario where we have a different frequency promise for every edge. We represent these promises with a weight function $\boldsymbol{w} \in \mathbb{N}^E$. An edge (u, v) is defined as admissible in the push-relabel algorithm if $\boldsymbol{\ell}(u) \geq \boldsymbol{\ell}(v) + \boldsymbol{w}(u, v)$. Following the logic of the above paragraph, suppose we have a promise that there exists a flow where every flow path has $\boldsymbol{w}$-weight at most $h = n^{1+o(1)}$, then we can guarantee that, when running the algorithm with maximum vertex label of $10h$, the algorithm will find a flow that routes a constant fraction of the demands. This yields the following theorem:

Theorem A.2.1 (Informal version of Theorem A.4.1)**.** *Given edge weights $\boldsymbol{w} \in \mathbb{N}^E$ and parameter h, the weighted push-relabel algorithm return a flow in $\widetilde{O}(m + h \cdot \sum_{e \in E} 1/\boldsymbol{w}(e))$ with the following guarantee: if there exists a flow $\boldsymbol{f}^*$ such that every flow path P in $\boldsymbol{f}$ has $\sum_{e \in P} \boldsymbol{w}(e) \leq h$, then the returned flow has value $\Omega(|\boldsymbol{f}^*|)$. In particular, if $\boldsymbol{f}^*$ is an α-approximate maximum flow, then the returned flow is a $O(\alpha)$-approximate maximum flow.*

The general idea of using a weight function to limit how often the algorithm touches various edges is inspired by a similar weighted variant of the Even-Shiloach trees [Ber17; PW20] that has been applied to dynamic shortest paths. A more detailed comparison of our push-relabel algorithm and the standard version and a discussion of possible future improvements for sparse graphs are given in Section A.4.2.

The Maximum Flow Algorithm. To this end, we say that a weight function $\boldsymbol{w}$ satisfies the *path-weight requirement* if there exists an $1/n^{o(1)}$-approximate maximum flow $\boldsymbol{f}$ such that every flow path P in $\boldsymbol{f}$ has $\sum_{e \in P} \boldsymbol{w}(e) \leq h = n^{1+o(1)}$. Our main

[5] The reason it is not technically true is that even though flow paths in the original graph use at most k infrequent edges, this is no longer true in the residual graph. But letting $\boldsymbol{f}_{\text{early}}$ be the flow already computed by push-relabel, note that the residual graph only differs from the original one by edges in $\boldsymbol{f}_{\text{early}}$, so if paths in the residual graph use significantly more infrequent edges than paths in the original graph, this implies that $\boldsymbol{f}_{\text{early}}$ is itself using many infrequent edges, and hence has a large value. In the technical exposition, we show that either $\boldsymbol{f}_{\text{early}}$ sends a constant fraction of the supply (so the algorithm can terminate), or paths in the residual graph are relatively similar to those in the original graph and hence have few infrequent edges.

technical contribution is showing how to compute a weight function $\boldsymbol{w}$ that satisfies the path-weight requirement and has $\sum_{e\in E} 1/\boldsymbol{w}(e) = n^{1+o(1)}$. Given this, by applying Theorem A.2.1, we immediately obtain a maximum flow algorithm with running time $n^{2+o(1)}$. (Recall that in directed graphs, an approximate flow algorithm immediately implies an exact algorithm.) In the remainder of this overview we explain how to get this weight function.

A.2.2 Examples of Good Weight Functions

Directed Acyclic Graphs. Let us consider the simplest directed graph: a directed acyclic graph (DAG). We know that a DAG admits a topological order $\boldsymbol{\tau} \in [n]^V$ such that $\boldsymbol{\tau}_v > \boldsymbol{\tau}_u$ for each edge (u, v). This topological order also gives us the desired weight function: if we set $\boldsymbol{w}(u, v) \coloneqq \boldsymbol{\tau}_v - \boldsymbol{\tau}_u$, then not only flow paths on the maximum flow, but any path in the DAG will have weight at most n. Moreover, it is easy to see that $\sum_{e\in E} 1/\boldsymbol{w}(e) = O(n \log(n))$, because the sum of weights incident to a specific vertex v forms a harmonic series and is hence $O(\log(n))$. Plugging this into Theorem A.2.1 yields a remarkably simple $\widetilde{O}(n^2)$-time algorithm for computing a $O(1)$-approximate flow in a DAG using only classical flow techniques (Corollary A.4.9).

General Graphs Given Maximum Flow. The analysis of the DAG case also shows the *existence* of a good weight function in general graphs: take any integral maximum flow, the support of which after cycle cancellation forms a DAG, and then assign weights as above to this support and assign large weight (e.g. $100n$) to all other edges. Of course, this weight function requires computing a maximum flow and, hence, is not useful for us. We will show another construction of good weight function based on a directed expander hierarchy.

A.2.3 Basic Facts About Expanders

In order to describe the directed expander hierarchy, we review some basic properties of expanders.

Definition A.2.2 (Directed expander)**.** Consider a directed, unweighted graph $G = (V, E)$. For any set of vertices $S \subseteq V$, we define $\mathrm{vol}(S) \coloneqq \sum_{v\in S} \deg(v)$, where $\deg(v)$ counts both in- and out-edges incident to v. We say that cut $\emptyset \neq S \subsetneq V$ is *ϕ-sparse* if $\min\{|E(S, \overline{S})|, |E(\overline{S}, S)|\} < \phi \cdot \min\{\mathrm{vol}(S), \mathrm{vol}(\overline{S})\}$, where $\overline{S} \coloneqq V \setminus S$. We say that a graph G is a *ϕ-expander* if it contains no ϕ-sparse cuts.

One should think of the ϕ parameter above as being $1/n^{o(1)}$. We also modify the above definitions to apply with respect to an edge set $F \subseteq E$, often referred to as *terminal* edges. In particular, define $\deg_F(v)$ to be the number of edges in F incident to v and $\mathrm{vol}_F(S) = \sum_{v\in S} \deg_F(v)$; we say a cut S is *ϕ-sparse* with respect to F if $\min\{|E(S, \overline{S})|, |E(\overline{S}, S)|\} < \phi \cdot \min\{\mathrm{vol}_F(S), \mathrm{vol}_F(\overline{S})\}$; we say that G is a *ϕ-expander* with respect to F if G contains no ϕ-sparse cuts with respect to F.

To handle graphs that are not strongly connected, it is useful to define a notion of expansion that applies separately to every strongly connected component (SCC). Given a set of terminal edges F, we say that F is *ϕ-expanding* in G if every SCC of G is a ϕ-expander with respect to F.[6]

Expanders are nice to work with in the context of flow problems because they admit a low-congestion flow between any sets of sources/sinks. This also generalizes to a terminal set F.

Fact A.2.3 (Proved in Lemma A.5.12)**.** *Let $G = (V, E)$ be a ϕ-expander with respect to a terminal set $F \subseteq E$. Consider any flow-instance with supply/demand $\mathbf{\Delta}, \mathbf{\nabla}$ such that $\|\mathbf{\Delta}\|_1 = \|\mathbf{\nabla}\|_1$ with all supply/demand on terminal edges; formally, this means that for every vertex v, $\mathbf{\Delta}(v) \leq \deg_F(v)$ and $\mathbf{\nabla}(v) \leq \deg_F(v)$. Then, there exists a flow $\boldsymbol{f}$ in G that routes all the supply/demand and has the following properties: **1)** $\boldsymbol{f}(e) = O(\log(n)/\phi)$ for every $e \in E$ and **2)** Every flow path in $\boldsymbol{f}$ uses at most $O(\log(n)/\phi)$ edges in F.*

A standard approach to dealing with a general undirected graph $G = (V, E)$ is to decompose it into a hierarchy of expanders [PT07; GRST21]; in this paper, we propose an analogous hierarchy for directed graphs. Let us first consider a single expander decomposition of a directed graph G, formalized in [BPS20]. In any directed graph G, it is possible to find a set of "back edges" B such that every strongly connected component (SCC) of $G \setminus B \coloneqq (V, E \setminus B)$ is a ϕ-expander and $|B| = \widetilde{O}(\phi m)$.[7] If we imagine a topological sort of the SCCs in $G \setminus B$, then the above partition effectively decomposes E into three edges types:

1. Edges inside SCCs of $G \setminus B$, which we denote as X_1.
2. Edges (u, v) between different SCCs of $G \setminus B$ that go forward in the topological ordering. We will denote these as D, which stands for DAG edges.
3. Edges in B, which may go backward in the topological ordering. We denote these as X_2.

Put succinctly, X_1 is ϕ-expanding in $G \setminus X_2$. If X_2 happens to be ϕ-expanding in G, then the expander hierarchy is complete; if not, we need to add a level to the hierarchy. We can again perform expander decomposition with respect to X_2 to compute an even smaller set of edges X_3 such that every SCC of $G \setminus X_3$ is a ϕ-expander with respect to $X_2 \setminus X_3$. Let us assume, for simplicity, that $X_3 \subseteq X_2$; then, to maintain a partition, we replace X_2 with $X_2 \setminus X_3$, and we now have a partition

[6] More precisely, each SCC is a ϕ-expander with respect to the volume induced by F.

[7] To see this existentially, imagine the algorithm that repeatedly finds ϕ-sparse cuts in the graph, adds the (the sparser direction of) cut edges to B, and recurses on both sides of the cut. This clearly results in a desired expander decomposition. The size of B can be bounded by a simple charging argument: Every time we find a sparse cut, we can charge the cut edges to the smaller side of the cut. Since a vertex can be in the smaller side of the cut at most $O(\log n)$ times, the bound of $\widetilde{O}(\phi m)$ follows.

of the edge set $E = D \cup X_1 \cup X_2 \cup X_3$. If X_3 is ϕ-expanding in G then the expander hierarchy is complete; otherwise we define a new set X_4 in the same manner. We now define the hierarchy more formally (see also Figure A.1 in Section A.5).

Definition A.2.4. A partition $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of the edges is a *ϕ-expander hierarchy* if D is acyclic and for every $i \in [\eta]$, X_i is ϕ-expanding in $G \setminus X_{>i}$; that is, all SCCs of $G \setminus X_{>i}$ are ϕ-expanders with respect to X_i, where $X_{>i} = X_{i+1} \cup \cdots \cup X_\eta$. Note that X_η must be ϕ-expanding in G.

While several variants of expander hierarchies have been previously used in *undirected* graphs [Räc02; PT07; GRST21], we believe ours is the first paper to apply them to directed graphs. The *existence* of the directed expander hierarchy below follows from generalizing the construction by [PT07] in undirected graphs. As we will discuss later, however, our construction is entirely different from previous approaches in undirected graphs.

Fact A.2.5. *Given any directed graph $G = (V, E)$ and $\phi \leq 1/\text{polylog}(n)$, there exists an expander hierarchy $\mathcal{H} = (D, X_1, ..., X_\eta)$ such that*

1. $|X_i| = \widetilde{O}(m\phi^{i-1})$.
2. *The total number of levels is around* $\log_{1/\phi}(m) = O(\log(n))$.

Note that the first property implies the second.

A.2.4 Directed Expander Hierarchy Implies Good Weight Function

The primary technical challenge lies in computing the expander hierarchy of Fact A.2.5. But first, in this section, we will show that once we compute such a hierarchy, it implies a good weight function that we can plug into Theorem A.2.1.

Simple Expander. Let us first consider the very simple case that the entire graph G is a ϕ-expander (for some $\phi = 1/n^{o(1)}$). In this case, we simply set $\boldsymbol{w}(e) = n$ for all $e \in E$. Note that $\sum_e 1/\boldsymbol{w}(e) = O(n)$; all that remains is to show that $\boldsymbol{w}$ satisfies the path-weight requirement. Let $\boldsymbol{f}^*$ be the actual maximum flow. The flow $\boldsymbol{f}^*$ itself may have long flow paths, but we will use the expansion of G to *shortcut* $\boldsymbol{f}^*$ while only paying a small overhead in congestion. By Fact A.2.3, there exists a flow $\boldsymbol{f}$ such that $\boldsymbol{f}$ routes the same supply/demand as $\boldsymbol{f}^*$, $\boldsymbol{f}$ has congestion $\widetilde{O}(1/\phi)$, and every flow path in $\boldsymbol{f}$ contains $\widetilde{O}(1/\phi)$ edges. Scaling $\boldsymbol{f}$ down by a $\widetilde{O}(\phi)$-factor thus yields a $\widetilde{\Omega}(1/n^{o(1)})$-approximate flow where every flow path P has weight $\boldsymbol{w}(P) = \widetilde{O}(n/\phi) = n^{1+o(1)}$. Note that the algorithm never explicitly computes $\boldsymbol{f}$; rather, we simply use its existence to argue that $\boldsymbol{w}$ is a good weight function.

DAG of Expanders. We now consider a slightly more general case, where every SCC of G is a ϕ-expander, but there can be DAG edges between the SCCs. This corresponds to a one-level expander hierarchy $\mathcal{H} = (D, X_1)$, where X_1 contains the edges inside SCCs of G, and D contains the inter-component edges. We say that a topological order $\boldsymbol{\tau}$ *respects* the SCCs of G if it has the following properties:

- For every edge $(u, v) \in D$, we have $\boldsymbol{\tau}_u < \boldsymbol{\tau}_v$.
- For every SCC C of G, the set $\boldsymbol{\tau}(C) \coloneqq \{\boldsymbol{\tau}_v : v \in C\}$ is contiguous; in other words, it contains precisely the set of numbers between $\boldsymbol{\tau}_{\min}(C) \coloneqq \min_{v \in C} \boldsymbol{\tau}_v$ and $\boldsymbol{\tau}_{\max}(C) \coloneqq \max_{v \in C} \boldsymbol{\tau}_v$.

It is easy to see that such a respecting $\boldsymbol{\tau}$ exists. We now define $\boldsymbol{w}(u, v) = |\boldsymbol{\tau}_v - \boldsymbol{\tau}_u|$. Not that if u, v are in the same SCC C, then $\boldsymbol{w}(u, v) \leq |C|$. Since the weight function is defined by a topological ordering we have that $\sum_e 1/\boldsymbol{w}(e) = O(n \log(n))$. The analysis is exactly the same as for the case when G is a DAG.

We now show that $\boldsymbol{w}$ satisfies the path-weight requirement. Let $\boldsymbol{f}^*$ be the maximum flow. As before, we start by shortcutting $\boldsymbol{f}^*$ inside each expander. Formally, for every component C of X_1, we apply Fact A.2.3 to the following flow instance: for every flow path P in $\boldsymbol{f}^*$, we add one unit of supply to the first vertex in $P \cap C$ and one unit of demand to the last vertex in $P \cap C$. Let $\boldsymbol{f}$ be the flow resulting from shortcutting $\boldsymbol{f}^*$ inside every SCC C. Note that $\boldsymbol{f}$ incurs a congestion of $1/\phi$ and that for every flow path P in $\boldsymbol{f}$, $|P \cap C| = \widetilde{O}(1/\phi)$.

We now argue that every flow path P in $\boldsymbol{f}$ has $\boldsymbol{w}(P) = \widetilde{O}(n/\phi)$. First, consider the weight of $X_1 \cap P$, i.e. the intra-component edges. For any component C, $P \cap C$ contains $\widetilde{O}(1/\phi)$ edges, each of weight at most $|C|$, so $\boldsymbol{w}(P \cap C) = \widetilde{O}(|C|/\phi)$; summing over all components yields weight $\widetilde{O}(n/\phi)$. For the inter-component edges on P, since the topological labels on these edges are monotonically increasing, it is easy to see that their total weight contribution is $O(n)$.

Two-Level Expander Hierarchy. The next slightly more general case is when the edges of G can be partitioned into a two-level expander hierarchy (D, X_1, X_2): X_1 contains edges inside SCCs of $G \setminus X_2$, and each of these SCCs is a ϕ-expander; D contains edges between SCCs of $G \setminus X_2$; finally, X_2 is expanding in G. This two-level hierarchy is far from the general case because of our assumption that X_2 is expanding in G; nonetheless, this special case will already contain all of our main ideas for proving that an expander hierarchy implies a good weight function.

The weight function is exactly the same as the previous one: we compute a topological order $\boldsymbol{\tau}$ that respects the SCCs of $G \setminus X_2$ and we set $\boldsymbol{w}(u, v) = |\boldsymbol{\tau}_v - \boldsymbol{\tau}_u|$. Since $\boldsymbol{w}$ is still based on a topological ordering, we again get $\sum_{e \in E} 1/\boldsymbol{w}(e) = O(n \log(n))$. All that remains is to show that $\boldsymbol{w}$ satisfies the path-weight requirement. To do so, we use the following claim:

Claim A.2.6. *Let $\boldsymbol{f}^*$ be the optimal maximum flow. There exists a flow $\boldsymbol{f}$ routing the same supply/demand as $\boldsymbol{f}^*$ does such that:*

1. *$\boldsymbol{f}$ has congestion $\widetilde{O}(1/\phi)$. (Actually we get* $\mathrm{cong}(f) = 1 + \frac{1}{\log(n)}$*, but $\widetilde{O}(1/\phi)$ is good enough.)*

2. *Every flow path in $\boldsymbol{f}$ contains $\widetilde{O}(1/\phi)$ edges from X_2.*

3. *Let P be any flow path in $\boldsymbol{f}$. For every SCC C of $G \setminus X_2$, we have $|P \cap C| = \widetilde{O}(1/\phi)$. (Recall that the SCCs of $G \setminus X_2$ are precisely the SCCs in which X_1 is ϕ-expanding.)*

Before proving this claim, let us see why it implies that $\boldsymbol{w}$ satisfies the path-weight requirement. Scaling $\boldsymbol{f}$ down by $\mathrm{cong}(\boldsymbol{f}) = \widetilde{O}(1/\phi)$ we get a feasible approximate flow, as desired. Consider any flow path P in $\boldsymbol{f}$. The path P contains at most $\widetilde{O}(1/\phi)$ edges from X_2, each with weight at most n, so the total weight contribution of $X_2 \cap P$ is $\widetilde{O}(n/\phi)$. For edges that belong to an SCC of $G \setminus X_2$, the analysis is exactly the same as for a DAG of expanders: each component C contributes $\widetilde{O}(|C|/\phi)$ weight to path P, for a total of $\widetilde{O}(n/\phi)$. Finally, consider the DAG edges in D. For any subpath of P that is disjoint from X_2, all the edges in D are increasing in terms of the $\boldsymbol{\tau}$ values, so the total weight of D-edges in such a subpath is $O(n)$. Every edge in X_2 can then go back to the beginning of the topological order, but since there are only $\widetilde{O}(1/\phi)$ edges in $P \cap X_2$, the total contribution of $P \cap D$ is $\widetilde{O}(n/\phi)$. We now sketch a proof of Claim A.2.6.

Proof Sketch of Claim A.2.6. Recall that we are assuming a two-level hierarchy where X_2 is expanding in G. By Fact A.2.3, we can thus reroute $\boldsymbol{f}^*$ to a new flow $\boldsymbol{f}_2$ such that $\boldsymbol{f}_2$ has congestion $c_2 = \widetilde{O}(1/\phi)$ and every flow path in $\boldsymbol{f}_2$ contains at most $\widetilde{O}(1/\phi)$ edges from X_2.

A Naïve Approach. We now need to further shortcut $\boldsymbol{f}_2$ so that it satisfies Property 3. Consider any SCC C of $G \setminus X_2$, and recall that by definition of expander hierarchy, C is a ϕ-expander with respect to X_1. The naïve way to shortcut the flow inside C is to repeat the procedure above: reroute flow from the first vertex in $P \cap C$ to the last, for every flow path P. There is, however, a subtle but significant issue with this approach. We are rerouting the flow $\boldsymbol{f}_2$ and not the original flow $\boldsymbol{f}^*$. Whereas $\boldsymbol{f}^*$ has congestion 1, the flow $\boldsymbol{f}_2$ already has congestion $c_2 = \widetilde{O}(1/\phi)$. For this reason, there could be a vertex $v \in C$ such that for every edge e entering v has a flow $\boldsymbol{f}_2(e) = c_2$ on it. As a result, the flow instance that we used to reroute C could have $\boldsymbol{\Delta}(v) \approx c_2 \deg(v) \approx \frac{1}{\phi}\deg(v)$, which exceeds the maximum specified by Fact A.2.3. We can still apply a scaled version of this fact, but the resulting edge congestion will then be $\widetilde{O}(c_2/\phi) = \widetilde{O}(1/\phi^2)$, instead of $\widetilde{O}(1/\phi)$. At first glance this might seem acceptable, since $1/\phi^2 = n^{o(1)}$. But for general graphs, the hierarchy might have as many as $\log_{1/\phi}(n)$ levels (see Fact A.2.5), and the naïve shortcutting approach above will multiply the congestion by $1/\phi$ per level, leading to an unacceptably high congestion of $\Omega(n)$.

All-to-All Rerouting With Less Demand Per Edge. To overcome this issue, we need a more careful shortcutting procedure. Consider again the flow $\boldsymbol{f}_2$ with congestion $c_2 = \widetilde{O}(1/\phi)$. We will show how to reroute $\boldsymbol{f}_2$ so that Property 3 of the claim is satisfied, while the congestion of the flow only increases to $c_2 \cdot (1+1/\log(n))$. Consider any SCC C of $G \setminus X_2$. Let $\mathcal{P}_{\text{short}}$ contain all flow paths P of $\boldsymbol{f}_2$ for which $|P \cap C| \leq 2k$, where k is a parameter we will later set to $\widetilde{O}(1/\phi)$.[8] Let $\mathcal{P}_{\text{long}}$ contain all other flow paths. Note that there is no need to reroute the flow paths of $\mathcal{P}_{\text{short}}$, as they already satisfy Property 3.

We define the following flow instance for rerouting $\mathcal{P}_{\text{long}}$. For every $P \in \mathcal{P}_{\text{long}}$, let P_{early} contains the first k vertices of P and P_{late} the last k. We reroute from all of P_{early} to all of P_{late}, which will allow us to place less supply/demand on every individual vertex. Formally, we add supply $1/k$ to every vertex in P_{early} and demand $1/k$ to every vertex in P_{late}. Since $\text{cong}(\boldsymbol{f}_2) = c_2$, the supply/demand on every vertex is now at most c_2/k. Therefore, applying (the scaled version of) Fact A.2.3, we get a flow $\boldsymbol{f}'$ with short flow paths and congestion $(c_2/k) \cdot \widetilde{O}(1/\phi)$. To reroute $\mathcal{P}_{\text{long}}$, we must combine flow $\boldsymbol{f}'$ with the flow along P_{early} and P_{late}, as the new flow must use P_{early} and P_{late} to reach all the sources and sinks on these segments. We now bound the overall congestion of the resulting flow $\boldsymbol{f}$. Any edge $e \in C$ includes at most $\boldsymbol{f}_2(e) \leq c_2$ from the parts of $\boldsymbol{f}_2$ that have not been rerouted, which includes all the flow from $\mathcal{P}_{\text{short}}$, as well all the flow from the early and late segments of each path in $\mathcal{P}_{\text{long}}$. Also, e gets an additional $(c_2/k) \cdot \widetilde{O}(1/\phi)$ units of flow from the rerouting. Together, this results in $\boldsymbol{f}(e) \leq c_2(1 + 1/k \cdot \widetilde{O}(1/\phi))$. Setting k to a large enough $\text{polylog}(n)/\phi$ yields $\boldsymbol{f}(e) \leq c_2(1+1/\log(n))$. Since there are $O(\log(n))$ levels in the expander hierarchy of Fact A.2.5, the congestion at the final level will still be $O(c_2) = \widetilde{O}(1/\phi)$. □

Generalizing to a Multi-Level Expander Hierarchy. Let us now consider a general graph G, which we know admits a multi-level expander hierarchy $\mathcal{H}$ as in Fact A.2.5. We can obtain a good weight function $\boldsymbol{w}$ using the same tools as in the simpler two-level hierarchy above.

First, let us say a topological order $\boldsymbol{\tau}$ respects $\mathcal{H} = (D, X_1, ..., X_\eta)$ if for every i, the $\boldsymbol{\tau}$ labels are contiguous in every SCC of $G \setminus X_{>i}$, and for every DAG edge $(u, v) \in D$ we have $\boldsymbol{\tau}_u < \boldsymbol{\tau}_v$. It is easy to construct such a topological order by going from the top to the bottom of the hierarchy, and computing SCCs in each $G \setminus X_{>i}$. We then define our weight function as $\boldsymbol{w}(u, v) = |\boldsymbol{\tau}_u - \boldsymbol{\tau}_v|$. Since $\boldsymbol{w}$ is defined by a topological order, we again have $\sum_e 1/\boldsymbol{w}(e) = O(n \log(n))$.

To prove that $\boldsymbol{w}$ satisfies the path-weight requirements we prove that there exists an approximate maximum flow $\boldsymbol{f}$ such that for every X_i and every SCC C of $G \setminus X_{>i}$, the flow $\boldsymbol{f}$ uses $\widetilde{O}(1/\phi)$ edges in $X_i \cap C$. It is easy to show that such a flow $\boldsymbol{f}$ satisfies the path-weight requirement, and we can show that such a flow $\boldsymbol{f}$

[8] Note that P might enter and leave C multiple times, but we can still consider the first (or last) k vertices of $P \cap C$.

exists by using the careful rerouting procedure above starting at the top level, then the second-highest level, and so on.

Remark A.2.7. The flow rerouting above is needed for analysis only. The algorithm never computes $\boldsymbol{f}$; instead, its existence is enough to prove that $\boldsymbol{w}$ satisfies the path-weight requirement. All the algorithm does is to find a hierarchy-respecting topological order $\boldsymbol{\tau}$ and set $\boldsymbol{w}(u,v) = |\boldsymbol{\tau}_v - \boldsymbol{\tau}_u|$.

A.2.5 Constructing the Directed Expander Hierarchy

As far as we know, our paper is the first to define a directed expander hierarchy. We first contrast our hierarchy with existing work in *undirected* graphs.

Previous Work: Undirected Expander Hierarchy. The definition of our hierarchy can be thought of as a generalization of the undirected hierarchies of [PT07]. The problem, however, is that [PT07] relies on a slow polynomial-time algorithm for constructing the hierarchy. It is possible to use techniques from [RST14] to efficiently construct the hierarchy in a top-down manner, but this requires solving a max-flow problem at each level, which we cannot afford to do as we are trying to develop our own efficient combinatorial max-flow algorithm. We thus develop an entirely different bottom-up construction.

A more recent paper of [GRST21] shows a different undirected expander hierarchy that admits a very efficient bottom-up construction. Their construction is based on boundary-linked expanders, which allow for low-congestion routing between the boundary edges. In *undirected* graphs, we observed that we could have naturally defined a good weight function from their hierarchy. Unfortunately, a decomposition into boundary-linked expanders does not exist for *directed graphs*. In undirected graphs, an expander decomposition has a small number (i.e., $\widetilde{O}(\phi m)$) of boundary edges, which is why boundary-linkedness is possible. In directed graphs there can be arbitrarily many boundary edges, because even if a cut $E(S, \overline{S})$ is sparse, there may still be $\Omega(m)$ edges in the other direction $E(\overline{S}, S)$.

Our Construction. We now give an overview of our framework for constructing the expander hierarchy $\mathcal{H}(D, X_1, ..., X_\eta)$ of Fact A.2.5. We proceed in a bottom-up fashion. The first step is to compute a set of edges X_2 such that $|X_2| = \widetilde{O}(\phi m)$ and all SCCs of $G \setminus X_2$ are ϕ-expanders; the edges inside these SCCs then become X_1. Loosely speaking, we can compute X_2 by repeatedly computing a ϕ-sparse cut and recursing on both sides (more details below). To construct the next level X_3 of the hierarchy, we again need to repeatedly find sparse cuts, but this time they need to be sparse with respect to X_2. Here, however, we encounter a potential issue: we may find a sparse cut $E(S, \overline{S})$ which is not a subset of X_2. As a result, when we move $E(S, \overline{S})$ to X_3, we will end up disturbing lower levels of the hierarchy. Unfortunately, there is no way to avoid this issue; in fact, depending on the choice of X_2, there might not even *exist* a cut that is sparse with respect to X_2 and whose

crossing edges are contained in X_2. This lack of nestedness poses a huge technical challenge which we discuss later, but let us bypass it for now and make the following unrealistic assumption:

Assumption A.2.8 (Unrealistic Nestedness Assumption)**.** Whenever we compute a cut $(S, \overline{S})$ that is ϕ-sparse with respect to some X_i, we are in the lucky case where $E(S, \overline{S}) \subseteq X_i$.

Given the assumption above, we can proceed to construct the whole hierarchy in a bottom-up fashion. The challenge now is to do so efficiently. As suggested above, finding the next edge set X_{i+1} requires repeatedly finding cuts that are sparse with respect to X_i. In fact, the whole construction can effectively be reduced to the following subroutine:

Sparse-Cut Subroutine. Given a graph $G = (V, E)$, a set of terminal edges $X_i \subseteq E$, a set of sources $X_{\boldsymbol{\Delta}} \subseteq X_i$, and a set of sinks $X_{\boldsymbol{\nabla}} \subseteq X_i$ with $|X_{\boldsymbol{\nabla}}| = |X_{\boldsymbol{\Delta}}|$, the algorithm must either:

1. find a flow $\boldsymbol{f}$ of congestion $\widetilde{O}(1/\phi)$, where all vertices in $X_{\boldsymbol{\Delta}}$ (resp., $X_{\boldsymbol{\nabla}}$) have one unit of supply (resp., demand) and $\boldsymbol{f}$ routes at least $|X_{\boldsymbol{\Delta}}|/2$ units of flow, or
2. find a cut that is ϕ-sparse with respect to X_i.

If we allow the nestedness assumption above, and have an efficient algorithm for the sparse-cut subroutine, then we can apply the standard approach of combining the subroutine with the celebrated cut-matching game framework [KRV06; Lou10] to either locate a sparse cut (without being given as input the $(X_{\boldsymbol{\Delta}}, X_{\boldsymbol{\nabla}})$ pair) in the graph or certify that it is an expander. By recursing on both sides of the sparse cut, we get an expander decomposition algorithm that computes X_{i+1}, and we can further ensure that the total number of invocations of the sparse-cut subroutine is $n^{o(1)}$ using ideas developed in [NS17; Wul17; NSW17].[9]

Sparse-Cut Subroutine: Level One. We now describe our implementation of the sparse-cut subroutine, which uses entirely new techniques. Let us start on the bottom level, where the set of terminal edges is $X_i = E$. At this level, the subroutine can easily be done in $\widetilde{O}(m/\phi)$ time using existing techniques (see e.g. [HRW17; SW19]), which we quickly review. The algorithm is quite simple: we run regular (non-weighted) push-relabel to send flow from $X_{\boldsymbol{\Delta}}$ to $X_{\boldsymbol{\nabla}}$, except that we allow edges to have capacity up to $\text{polylog}(n)/\phi$, and we impose a maximum vertex label of $h = \widetilde{O}(1/\phi)$; this artificial maximum might prevent push-relabel from finding a maximum flow. Let $\boldsymbol{f}$ be the flow computed by push-relabel. There are two cases

[9] Similar ideas were previously applied to *directed* expander decomposition/pruning in [BPS20; HKPW23]. We make particular use of the algorithmic framework established by [HKPW23] later in the paper to handle the unrealistic nestedness assumption.

to consider. If $\boldsymbol{f}$ sends at least $|X_{\boldsymbol{\Delta}}|/2$ flow, we are done. If not, let $G_{\boldsymbol{f}}$ be the remaining residual graph, and note that since $\boldsymbol{f}$ has small value, there must exist some $s \in X_{\boldsymbol{\Delta}}$ and some $t \in X_{\boldsymbol{\nabla}}$ with $\boldsymbol{\ell}(s) = h = \widetilde{O}(1/\phi)$ and $\boldsymbol{\ell}(t) = 0$.

Rather than working directly with the labels $\boldsymbol{\ell}$, our algorithm computes a new labelling $\boldsymbol{d}$, where $\boldsymbol{d}(v) \coloneqq \mathrm{dist}_{G_{\boldsymbol{f}}}(v, t)$. By the admissibility property of push-relabel, we have $\boldsymbol{d}(s) \geq h$. Now, for any k, define $V_k \coloneqq \{v \in V \mid \boldsymbol{d}(v) = k\}$ and $V_{\geq k} \coloneqq \{v \in V \mid \boldsymbol{d}(v) \geq k\}$. We refer to cuts $(S, \overline{S})$ of the form $S = V_{\geq k}$ as a *level cut*. We will show that one of the level cuts is a sparse cut in $G_{\boldsymbol{f}}$. For the full proof we need to show that one of the level cuts is sparse in the original graph G, but the proof is essentially the same: loosely speaking, since we set edge capacities to be $\mathrm{polylog}(n)/\phi$, the flow $\boldsymbol{f}$ saturates at most $O(\phi|X_{\boldsymbol{\Delta}}|/\mathrm{polylog}(n))$ edges, which is so few that they have minimal effect on the sparseness of a level cut.

To see that one of the level cuts in $G_{\boldsymbol{f}}$ is sparse, we use a so-called ball-growing argument. Consider some level cut $S = V_{\geq k}$, and note that since $\boldsymbol{d}$ corresponds to distances, every edge in G' leaving $V_{\geq k}$ goes to V_{k-1}. So if cut S is non-sparse, then there are many edges from $V_{\geq k}$ to V_{k-1}, and in particular $\mathrm{vol}(V_{\geq k-1}) \geq \mathrm{vol}(V_{\geq k}) \cdot (1 + \phi)$. Thus, there can be at most $\log_{1+\phi}(m) = \widetilde{O}(1/\phi)$ non-sparse layers, so as long as we set h large enough, we can ensure that over half the level cuts are sparse.

Sparse-Cut Subroutine: Level Two. Let us now consider the case where we have already constructed the first level of the hierarchy $\mathcal{H} = (D, X_1, X_2)$. To construct the next layer, we need to solve the sparse-cut subroutine with respect to terminal edges X_2. This simple case will once again contain most of our main ideas for finding a sparse cut with respect to a general X_i.

We can no longer directly use a ball-growing argument. In the simple case above, the crux of the argument was that the edges crossing any non-sparse level cut $S = V_{\geq k}$ get added to the volume of $V_{\geq k-1}$, which guarantees that $\mathrm{vol}(V_{\geq k})$ increased multiplicatively as we move from $k = h$ to $k = 0$. The problem is that for the second level of the hierarchy, sparseness is defined with respect to vol_{X_2}, but the edges crossing a non-sparse cut $S = V_{\geq k}$ might not belong to X_2, so $\mathrm{vol}_{X_2}(V_{\geq k})$ might not change at all across levels. In order to use ball-growing to argue that there exists a sparse level cut, we will need to reassign vertex levels in such a way that there exist many level cuts whose edges come primarily from X_2.

The key idea is to use the weighted push-relabel algorithm, where the weight $\boldsymbol{w}$ will be based on the incomplete hierarchy we have already built. In particular, let $\boldsymbol{\tau}$ be a topological order that respects the SCCs of $G \backslash X_2$ and let $\boldsymbol{w}(u, v) = |\boldsymbol{\tau}_u - \boldsymbol{\tau}_v|$. We will now run the weighted push-relabel algorithm up to maximum label $h = \widetilde{O}(n/\phi)$; by Theorem A.2.1 the runtime is $O(m + h\sum_{e \in E} 1/\boldsymbol{w}(e)) = O(m + hn\log(n)) = n^{2+o(1)}$. Let $\boldsymbol{f}$ be the flow computed by weighted push-relabel. If $\boldsymbol{f}$ sends at least $|X_{\boldsymbol{\Delta}}|/2$ flow, then we are done. Otherwise, we once again have vertices $s \in X_{\boldsymbol{\Delta}}$ and $t \in X_{\boldsymbol{\nabla}}$ with $\boldsymbol{\ell}(s) = h = \widetilde{O}(n/\phi)$ and $\boldsymbol{\ell}(t) = 0$. As before, define $\boldsymbol{d}(v) \coloneqq \mathrm{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}}}(v, t)$,

where $\text{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}}}$ is the shortest distance according to $\boldsymbol{w}$ in the residual graph. We know that $\boldsymbol{d}(s) \geq h$.

Now, for the sake of intuition, consider the simplistic case where $\boldsymbol{f}$ is empty, so the residual graph $G_{\boldsymbol{f}} = G$. As discussed above, to argue that there exists a level cut that is sparse with respect to X_2, we need there to be many level cuts whose edges come primarily from X_2. This might not be true under the current labelling $\boldsymbol{d}$ because of the presence of DAG edges, so we define a new weight function $\boldsymbol{w}'$, which is the same as $\boldsymbol{w}$ except that it sets the weight of all DAG edges to 0. We then define labeling $\boldsymbol{d}'(v) \coloneqq \text{dist}^{\boldsymbol{w}'}_{G_{\boldsymbol{f}}}(v, t)$. Even though $\boldsymbol{d}'(s) < \boldsymbol{d}(s)$, we argue that $\boldsymbol{d}'(s) \geq \boldsymbol{d}(s)/2 - n = \Omega(h)$, so we still have many levels. This follows from the fact that under the original weight function $\boldsymbol{w}$, the DAG edges in D always increase $\boldsymbol{\tau}$, so except for the initial increase from $\boldsymbol{\tau} = 0$ to $\boldsymbol{\tau} = n$, any further weight-contribution from D must be balanced by edges in X_1 and X_2 that move backward in the topological ordering; as a result, D can only account for around half the total weight of a path under $\boldsymbol{w}$, so $\boldsymbol{d}'(s) \gtrsim \boldsymbol{d}(s)/2$.

We thus have a distance labeling $\boldsymbol{d}'$ such that $\boldsymbol{d}'(s) = \Omega(h)$ and none of the level cuts contain any DAG edges (because they have weight 0). We now argue that most of the level cuts also do not contain any edges from X_1. Recall that the SCCs of X_1 are ϕ-expanders. Consider any SCC C of X_1; since for any edge $(u, v) \in C$ we have $\boldsymbol{w}'(u, v) = \boldsymbol{w}(u, v) \leq |C|$, the diameter of C under $\boldsymbol{w}'$ is at most $\widetilde{O}(|C|/\phi)$, so edges inside C are present in at most $\widetilde{O}(|C|/\phi)$ different level cuts. Therefore, in total there are at most $\widetilde{O}(n/\phi)$ level cuts containing edges from X_1, and if we set $h = \widetilde{O}(n/\phi)$ large enough then there will be $\Omega(h)$ level cuts that contain exclusively edges from X_2. We can now use a standard ball-growing argument to argue that one of these level cuts is sparse with respect to X_2.

Recall that we made the simplifying assumption that $G_{\boldsymbol{f}} = G$. In reality, weighted push-relabel might compute some initial flow $\boldsymbol{f}$, so $G_{\boldsymbol{f}} \neq G$. We now argue that we can still find a cut in the residual graph $G_{\boldsymbol{f}}$ that is sparse with respect to X_2, which as already discussed, also yields a sparse cut in G. We start by again setting $\boldsymbol{w}'$ to have weight 0 on all edges in D except the residual edges of flow $\boldsymbol{f}$, and we define distance function $\boldsymbol{d}'$ accordingly. The residual edges of flow $\boldsymbol{f}$ have a small contribution,[10] so we ignore them for this overview; as a result, we again have that $\boldsymbol{d}'(s) = \Omega(h)$ and level cuts that contain no edges in D.

Dealing with the edges of X_1 is trickier. Consider a SCC C of X_1. The problem is that if the flow $\boldsymbol{f}$ saturated some edges in C, then those edges are reversed in $G_{\boldsymbol{f}}$, so C might no longer be an expander, and hence might have high diameter. The crux of our analysis is to argue that, as the value of flow $\boldsymbol{f}$ is relatively small, it does not impact the average expander C by too much.

To argue this, a natural idea is to apply the expander pruning argument (see e.g., [SW19; BPS20; HKPW23]). In particular, if $\boldsymbol{f}$ saturates σ *edges* in C, then

[10] The *residual edges* are those edges which we sent flow along, and are then reversed in the residual graph. Our weighted push-relabel algorithm will guarantee that each augmenting path it finds is of $\boldsymbol{w}$-length $O(h)$, so the contribution of these edges to the level cuts is not too much.

there exist a pruned set $P_C \subseteq C$ such that P_C has small size $\widetilde{O}(\sigma/\phi)$ and $C \setminus P_C$ is still a $\Omega(\phi)$-expander. Thus, since $C \setminus P_C$ has small diameter, its edges are once again present in only a minority of level cuts, so the remaining level cuts only contain edges from X_2 and from the pruned parts P_C. As long as the pruned parts are small we can argue that their impact is minimal, and thus most level cuts contain edges primarily from X_2. Again, standard ball-growing techniques proves that one of the remaining level cuts is sparse with respect to X_2. Unfortunately, the standard expander pruning technique does not give a small enough pruned set P_C.

Technical Highlight: Path-Reversal Pruning. The remaining challenge is in arguing that the pruned set P_C is small. Let R be some flow path of $\boldsymbol{f}$ that goes through C (R for reversed path). Since $\boldsymbol{f}$ is relatively small, the number of such flow paths is also small. The problem is that $|R|$ can contain many edges, so if we apply standard expander pruning by simply deleting all of R, the resulting pruned set P_C will be too large.

To overcome this challenge, we introduce a new technique, *path-reversal pruning*, which we believe might find other applications. Note that R is not actually deleted from the residual graph $G_{\boldsymbol{f}}$; instead, its edges are reversed. Reversing an entire path only changes the size of any directed cut by at most 1, and so intuitively it should not affect expansion by too much. We are able to show that from the perspective of pruning, reversing an entire path (no matter the length) has approximately the same impact as deleting a single edge. In particular, we prove that if we reverse σ different paths $R_1, ..., R_\sigma$, then there exists a pruned set P_C such that $C \setminus P_C$ is still an expander and the size of P_C is roughly σ/ϕ, rather than $\sum_{i=1}^{\sigma} |R_i|/\phi$ given by previous pruning guarantees. The technical details end up being quite different from standard pruning.

Remark A.2.9. Note that the algorithm itself never performs any pruning. All it does is: compute a flow $\boldsymbol{f}$ using weighted push-relabel, change the weight of the DAG edges to 0, compute new distance labels $\boldsymbol{d}'$ using Dijkstra's algorithm, and then check all the level cuts until it finds a sparse one. Pruning is used only in the analysis to argue that one of the level cuts is indeed sparse.

Edge Capacities. All of the analysis and expander decomposition tools generalize almost seamlessly to capacitated graphs. To make our weighted push relabel algorithm still efficient in capacitated graphs we use dynamic trees [ST83], similar to what is done for a standard push relabel [GT88].

A.2.6 Removing the Unrealistic Nestedness Assumption

Until now, we have assumed Assumption A.2.8 that when we compute a ϕ-sparse cut S with respect terminal edge set X_i, we always have $E(S, \overline{S}) \subseteq X_i$, i.e., the cut edges consist only of the terminal edges. Unfortunately, there are many counterexamples showing this assumption is impossible. Without Assumption A.2.8, the following

issue occurs: once a sparse cut S in $G[U]$ is found, our algorithm needs to further recurse on both sides S and $U \setminus S$, yet if the cut contains non-terminal edges, then we no longer have an expander hierarchy of $G[S]$ and $G[U \setminus S]$ from lower levels that our flow algorithm needs when performing the recursions.

In Section A.7 we address this problem. In particular, instead of fixing the i-th level expanding edges X_i once it is computed, we allow edges to be moved between different levels in the hierarchy to ensure nestedness. Similarly, our algorithm also moves between levels and may attempt to find further sparse cuts following edge movements. To modularize the analysis, we employ a data structure point-of-view that models these interactions between levels. We adapt the framework of [HKPW23] to maintain a single-level expander decomposition when edges are moving between levels. However, unlike the analysis of [HKPW23], our approach is not inherently *dynamic* in the sense that we do not exploit any local property of the weighted push-relabel algorithm we developed. Instead, our focus is on arguing that the total number of updates given to these data structures is small throughout the construction of the hierarchy. That is, in contrast to achieving a local and sublinear update time as in the dynamic graph algorithm literature and previous maximum flow algorithms, our data structure spends $n^{2+o(1)}$ time *per update*, which when combined with the analysis that there are only $n^{o(1)}$ updates results in the final running time. We defer a more detailed overview of our approach to Section A.7.1.

We acknowledge that our current construction (spanning more than 40 pages in Section A.7) seems overly involved (unlike the otherwise relatively simple algorithm parts of our paper) and we believe that with future developments of directed expander-related techniques this can be greatly simplified. We also emphasize that this step of avoiding non-nested cuts is the only reason why our algorithm is randomized[11] and has an inherent subpolynomial overhead.[12]

A.3 Preliminaries

General Notation. We use $\mathbb{N}$ to denote the set of *nonnegative* integers. Let $[k]$ for $k \in \mathbb{N}$ be $\{1, \ldots, k\}$, and in particular $[0] \coloneqq \emptyset$. For a collection of sets $\{S_i\}_{\ell \le i \le r}$ indexed by integers, let $S_{\le j} \coloneqq \bigcup_{\ell \le i \le j} S_i$ and $S_{\ge j} \coloneqq \bigcup_{j \le i \le r} S_i$, and define $S_{<j}$ and $S_{>j}$ analogously. We let $\boldsymbol{a} \le \boldsymbol{b}$ for vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ act entry-wise. For a vector $\boldsymbol{x} \in \mathbb{R}^U$ we may write $\boldsymbol{x}(S) \coloneqq \sum_{u \in S} \boldsymbol{x}(u)$ for $S \subseteq U$. We use $\mathbf{0}$ and $\mathbf{1}$ to denote the all-zero and all-one vectors whose dimensions shall be clear from context.

We say an event happens *with high probability* if it does with probability at least $1 - n^{-c}$ for an arbitrarily large (but fixed) constant $c > 0$.

[11] We also use a randomized cut-matching game from [KRV06; Lou10], but that can be easily replaced with a deterministic counterpart [BPS20].

[12] Technically speaking, most current directed expander decomposition algorithms run in almost-linear instead of near-linear time. However, with the recent work of [SP24] it seems promising that one can adopt their techniques in combination with our push-relabel algorithm to achieve a $\widetilde{O}(n^2)$ construction, at least if *assuming the unrealistic nestedness assumption.*

Graphs. Graphs in this paper are assumed to be directed. Unless explicitly stated to be *simple*, multi-edges are allowed. Let $G = (V, E)$ be a graph. For disjoint subsets $A, B \subseteq V$, let $E_G(A, B) \coloneqq \{(u, v) : u \in A, v \in B\}$. Let $\delta^+_G(v) \coloneqq E_G(\{v\}, V \setminus \{v\})$ and $\delta^-_G(v) \coloneqq E_G(V \setminus \{v\}, v)$ be the *outward* and *inward* edges incident to v. Let $\delta_G(v) \coloneqq \delta^+(v) \cup \delta^-(v)$. When clear from context, let $\overline{S}$ for $S \subseteq V$ be $V \setminus S$. For instance, we write $E_{G[U]}(S, \overline{S}) \coloneqq E_{G[U]}(S, U \setminus S)$ for $U \subseteq V$. Let $\overleftarrow{G} = (V, \overleftarrow{E})$ be G where all edges are reversed, i.e., $\overleftarrow{E} \coloneqq \{(v, u) : (u, v) \in E\}$. The *edge-vertex incidence matrix* $\boldsymbol{B}_G \in \{-1, 0, 1\}^{E \times V}$ of G is given by $\boldsymbol{B}_G(e, u) = 1$ and $\boldsymbol{B}_G(e, v) = -1$ for each $e = (u, v) \in E$ with all other entries set to zero.

A graph G is *strongly connected* if $E_G(S, \overline{S}) \neq \emptyset$ for every $\emptyset \neq S \subsetneq V$. A *strongly connected component* of G is a maximal strongly connected subgraph of G. Let $\mathrm{SCC}(G)$ denote the collection of strongly connected components of G. An edge set $F \subseteq E$ is a *separator* of G if no edge in F has both its endpoints in the same strongly connected components of $G \setminus F$.

Capacitated Graphs. We consider capacitated graphs $(G, \boldsymbol{c})$ with capacities $\boldsymbol{c} \in \mathbb{N}^E$. Unless stated otherwise, throughout this paper by standard capacity scaling (see Section A.9) we assume $\boldsymbol{c}(e) \leq n^2$ for all $e \in E$. For a subgraph $H \subseteq G$, we may overload notation and $(H, \boldsymbol{c})$ to denote a capacitated graph with capacities $\boldsymbol{c}$ restricted H. For $F \subseteq E$, let $\deg^+_{F,\boldsymbol{c}}(v) \coloneqq \sum_{e \in \delta^+(v) \cap F} \boldsymbol{c}(e)$, $\deg^-_{F,\boldsymbol{c}}(v) \coloneqq \sum_{e \in \delta^-(v) \cap F} \boldsymbol{c}(e)$, and $\deg_{F,\boldsymbol{c}}(v) \coloneqq \deg^+_{F,\boldsymbol{c}}(v) + \deg^-_{F,\boldsymbol{c}}(v)$ be the sum of capacities of edges in F incident to v. Let $\mathrm{vol}_{F,\boldsymbol{c}}(S) \coloneqq \sum_{v \in S} \deg_{F,\boldsymbol{c}}(v)$ for $S \subseteq V$. When G is clear from context, let $\deg_{\boldsymbol{c}}(v) \coloneqq \deg_{E,\boldsymbol{c}}(v)$ and $\mathrm{vol}_{\boldsymbol{c}}(S) \coloneqq \mathrm{vol}_{E,\boldsymbol{c}}(S)$. When the graph is unit-capacitated, i.e., $\boldsymbol{c} = \mathbf{1}$, we drop the subscript $\boldsymbol{c}$ in the above notation which recovers the standard definitions of degree and volume. For analysis it is oftentimes simpler to work with unit-capacitated graphs. Let $G^{\boldsymbol{c}}$ be G where each edge e is duplicated $\boldsymbol{c}(e)$ times. For $F \subseteq E$, let $F^{\boldsymbol{c}} \subseteq E(G^{\boldsymbol{c}})$ be the multi-subset of $E(G^{\boldsymbol{c}})$ that contains precisely the duplicates of edges in F. It is easy to see that the above definitions are equivalent in $(G, \boldsymbol{c})$ and $G^{\boldsymbol{c}}$. Let G^c for $c \in \mathbb{N}$ be $G^{c\mathbf{1}}$ and F^c be $F^{c\mathbf{1}}$.

Flows. A *flow instance* $\mathcal{I}$ is a tuple $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ where $G = (V, E)$ is a graph with edge capacities $\boldsymbol{c} \in \mathbb{N}^E$, $\boldsymbol{\Delta} \in \mathbb{R}^V_{\geq 0}$ is the source vector, and $\boldsymbol{\nabla} \in \mathbb{R}^V_{\geq 0}$ is the sink vector. Without stated otherwise, we further assume $\|\boldsymbol{\Delta}\|_1 \leq \|\boldsymbol{\nabla}\|_1$, i.e., $\mathcal{I}$ is a *diffusion* instance. When unspecified, we assume the graph is unit-capacitated, i.e., $\boldsymbol{c} = \mathbf{1}$. Consider a vector $\boldsymbol{f} \in \mathbb{Q}^E_{\geq 0}$.[13] The *absorption* of $\boldsymbol{f}$ is $\mathbf{abs}_{\boldsymbol{f}} \coloneqq \min\{\boldsymbol{B}_G^\top \boldsymbol{f} + \boldsymbol{\Delta}, \boldsymbol{\nabla}\}$, where the min operator is defined entry-wise. The *excess* of $\boldsymbol{f}$ is $\mathbf{ex}_{\boldsymbol{f}} \coloneqq \boldsymbol{B}_G^\top \boldsymbol{f} + \boldsymbol{\Delta} - \mathbf{abs}_{\boldsymbol{f}} = \max\{\boldsymbol{B}_G^\top \boldsymbol{f} + \boldsymbol{\Delta} - \boldsymbol{\nabla}, \mathbf{0}\}$. The *value*

[13] In general, flows in graphs can take real values on edges. However, our algorithms and analyses will always work with flows of rational values, and in particular restricting $\boldsymbol{f}$ to be in $\mathbb{Q}^E$ allows us to treat a fractional flow as an integral flow in the graph in which edges are duplicated, making our analyses cleaner.

of $\boldsymbol{f}$ is $|\boldsymbol{f}| \coloneqq \mathbf{abs}_{\boldsymbol{f}}(V) = \|\boldsymbol{\Delta}\|_1 - \mathbf{ex}_{\boldsymbol{f}}(V)$. Let $\boldsymbol{f}^{\text{out}} \coloneqq -\boldsymbol{B}_G^\top \boldsymbol{f}$ and so $\boldsymbol{f}^{\text{out}}(v)$ is the net flow going out of v. The vector $\boldsymbol{f}$ is a $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-*flow*, or simply a *flow*, if $\mathbf{0} \leq \mathbf{ex}_{\boldsymbol{f}} \leq \boldsymbol{\Delta}$. The *congestion* of $\boldsymbol{f}$ is $\text{cong}(\boldsymbol{f}) \coloneqq \|\boldsymbol{f}/\boldsymbol{c}\|_\infty$. A flow is *feasible* if $\boldsymbol{f} \leq \boldsymbol{c}$ or equivalently $\text{cong}(\boldsymbol{f}) \leq 1$. The flow $\boldsymbol{f}$ *routes* $\mathcal{I}$ (or routes the demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$) if $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1 = \|\boldsymbol{\nabla}\|_1$, and we say $\mathcal{I}$ is *routable* with congestion κ if $\text{cong}(\boldsymbol{f}) \leq \kappa$ for such a flow (or simply *routable* if $\kappa \leq 1$). Two flows $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are *equivalent* if they route the same demand, i.e., $\boldsymbol{B}_G^\top \boldsymbol{f}_1 = \boldsymbol{B}_G^\top \boldsymbol{f}_2$.

Fact A.3.1. *For any flow $\boldsymbol{f}$ and $S \subseteq V$ it holds that $\boldsymbol{\Delta}(S) = \mathbf{abs}_{\boldsymbol{f}}(S) + \boldsymbol{f}^{\text{out}}(S) + \mathbf{ex}_{\boldsymbol{f}}(S)$.*

Given a flow $\boldsymbol{f}$, the *residual graph* $G_{\boldsymbol{f}}$ contains for each $e = (u, v) \in E$ a *forward* edge $\overrightarrow{e} = (u, v)$ with capacity $\boldsymbol{c}_{\boldsymbol{f}}(\overrightarrow{e}) \coloneqq \boldsymbol{c}(e) - \boldsymbol{f}(e)$ if $\boldsymbol{c}_{\boldsymbol{f}}(\overrightarrow{e}) > 0$ and a *backward* edge $\overleftarrow{e} = (v, u)$ with capacity $\boldsymbol{c}_{\boldsymbol{f}}(\overleftarrow{e}) \coloneqq \boldsymbol{f}(e)$ if $\boldsymbol{c}_{\boldsymbol{f}}(\overleftarrow{e}) > 0$. For $F \subseteq E$, let $\overrightarrow{F} \coloneqq \{\overrightarrow{e} : e \in F\}$ and $\overleftarrow{F} \coloneqq \{\overleftarrow{e} : e \in F\}$. Let $\boldsymbol{\Delta}_{\boldsymbol{f}} \coloneqq \mathbf{ex}_{\boldsymbol{f}}$ and $\boldsymbol{\nabla}_{\boldsymbol{f}} \coloneqq \boldsymbol{\nabla} - \mathbf{abs}_{\boldsymbol{f}}$ be the *residual sources* and *residual sinks*. A source s with $\boldsymbol{\Delta}(s) > 0$ is *unsaturated* by $\boldsymbol{f}$ is $\boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0$; likewise, a sink t with $\boldsymbol{\nabla}(t) > 0$ is *unsaturated* if $\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$. Together this defines the residual flow instance $\mathcal{I}_{\boldsymbol{f}} = (G_{\boldsymbol{f}}, \boldsymbol{c}_{\boldsymbol{f}}, \boldsymbol{\Delta}_{\boldsymbol{f}}, \boldsymbol{\nabla}_{\boldsymbol{f}})$. An *augmenting path* is a path in $G_{\boldsymbol{f}}$ consisting of edges with positive residual capacities from an unsaturated source to an unsaturated sink. The following standard fact justifies the use of residual graphs.

Fact A.3.2. *For any feasible flow $\boldsymbol{f}$ of $\mathcal{I}$, it holds that if $\boldsymbol{f}'$ is a maximum flow of $\mathcal{I}_{\boldsymbol{f}}$ then $\boldsymbol{f} + \boldsymbol{f}'$ is a maximum flow of $\mathcal{I}$.*

A flow $\boldsymbol{f}$ is *integral* if $\boldsymbol{f} \in \mathbb{N}^E$. Otherwise, $\boldsymbol{f}$ is *fractional* and $\frac{1}{z}$-*integral* for $z \in \mathbb{N}$ such that $\boldsymbol{f} \in (\frac{1}{z} \cdot \mathbb{N})^E$. When $\boldsymbol{f}$ is $\frac{1}{z}$-integral, we often equivalently view it as an integral flow in the unit-capacitated $G^{(z \cdot \boldsymbol{c})}$ and decompose it into a collection of flow paths through the following standard fact. Let $\boldsymbol{f}_P$ for P a path in G be the flow that sends one unit of flow along P, i.e., $\boldsymbol{f}(e) = 1$ for all $e \in P$. While the capacitated perspective allows for faster algorithms, the unit-capacitated one is sometimes easier to work with for analysis, as demonstrated by, e.g., the following standard fact.

Fact A.3.3. *An integral flow $\boldsymbol{f}$ admits a path decomposition $\mathcal{P}_{\boldsymbol{f}} \coloneqq \{P_1, \ldots, P_{|\boldsymbol{f}|}\}$ such that $\boldsymbol{f}' \coloneqq \boldsymbol{f}_{P_1} + \cdots + \boldsymbol{f}_{P_{\boldsymbol{f}}}$ is equivalent to $\boldsymbol{f}$ and satisfies $\boldsymbol{f}' \leq \boldsymbol{f}$.*

The following equivalence between maximum flow and minimum cut is standard.

Fact A.3.4 (Max-flow min-cut theorem)**.** *For a flow instance $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ the maximum flow value is equal to*

$$\min_{S \subseteq V} \boldsymbol{c}(E_G(S, \overline{S})) + \boldsymbol{\Delta}(\overline{S}) + \boldsymbol{\nabla}(S).$$

The *maximum (s,t)-flow* or simply the *maximum flow* problem is to find a maximum $(\boldsymbol{\Delta}_s, \boldsymbol{\nabla}_t)$-flow with $\boldsymbol{\Delta}_s \coloneqq \infty \cdot \mathbf{1}_s$ and $\boldsymbol{\nabla}_t \coloneqq \infty \cdot \mathbf{1}_t$.

Weights and Distances. Consider some edge weights $\boldsymbol{w} \in \mathbb{N}^E$. Let $\text{dist}_G^{\boldsymbol{w}}(s,t)$ be the shortest (s,t)-distance in G with respect to $\boldsymbol{w}$. This is also referred to as the $\boldsymbol{w}$-*distance* between s and t in G. Let $\text{dist}_G^{\boldsymbol{w}}(S,T)$ for $S,T \subseteq V$ be $\min_{s\in S,t\in T} \text{dist}_G^{\boldsymbol{w}}(s,t)$. For any flow $\boldsymbol{f}$, we often extend $\boldsymbol{w}$ to assign the same weight $\boldsymbol{w}(e)$ to both $\overrightarrow{e}$ and $\overleftarrow{e}$ in $G_{\boldsymbol{f}}$ when referring to $\text{dist}_{G_{\boldsymbol{f}}}^{\boldsymbol{w}}(s,t)$. The *weight* of a flow is $\boldsymbol{w}(\boldsymbol{f}) \coloneqq \sum_{e\in E} \boldsymbol{w}(e)\boldsymbol{f}(e)$. The $\boldsymbol{w}$-*length* of a path P is $\sum_{e\in P} \boldsymbol{w}(e)$. For $F \subseteq E$, the F-*distance* and F-*length* are defined as the $\boldsymbol{w}_F$-distance and $\boldsymbol{w}_F$-length for $\boldsymbol{w}_F(e) = 1$ for $e \in F$ and $\boldsymbol{w}_F(e)$ for $e \in E \setminus F$.

Expanders. Consider first a strongly connected capacitated graph $(G,\boldsymbol{c})$ and vertex weights $\boldsymbol{\nu} \in \mathbb{R}^V_{\geq 0}$. A cut $\emptyset \neq S \subsetneq V$ is ϕ-*sparse with respect to* $\boldsymbol{\nu}$ *in* $(G,\boldsymbol{c})$ if $\min\{\boldsymbol{c}(E_G(S,\overline{S})), \boldsymbol{c}(E_G(\overline{S},S))\} < \phi \cdot \min\{\boldsymbol{\nu}(S), \boldsymbol{\nu}(\overline{S})\}$. We say that $\boldsymbol{\nu}$ is ϕ-*expanding* in $(G,\boldsymbol{c})$ if there is no ϕ-sparse cut in G with respect to $\boldsymbol{\nu}$. For G that is not necessarily strongly connected, we say that $\boldsymbol{\nu}$ is ϕ-*expanding* in $(G,\boldsymbol{c})$ if $\boldsymbol{\nu}$ restricted to $U \subseteq V$ is ϕ-expanding in $(G[U],\boldsymbol{c})$ for every strongly connected component U of G. An edge set $F \subseteq E$ is ϕ-*expanding* if $\deg_{F,\boldsymbol{c}}$ is ϕ-expanding in G. We may sometimes overload notation and say that F is ϕ-expanding in a subgraph $H \subseteq G$ if $\deg_{F,\boldsymbol{c}}$ restricted to $V(H)$ is ϕ-expanding in H. When the graph is unit-capacitated, i.e., when $\boldsymbol{c} = \mathbf{1}$, we may drop the vector $\boldsymbol{c}$ in the notation. A ϕ-*(pure)-expander* is a $(G,\boldsymbol{c})$ in which $E(G)$ is ϕ-expanding. For analysis of our algorithm, we often make use of the following equivalence between uncapacitated and capacitated expanders.

Fact A.3.5. *An edge set F is ϕ-expanding in $(G,\boldsymbol{c})$ if and only if $F^{\boldsymbol{c}}$ is ϕ-expanding in $G^{\boldsymbol{c}}$.*

Embedding. An *embedding* $\Pi_{H\to G}$ from $(H,\boldsymbol{c}_H)$ to $(G,\boldsymbol{c}_G)$ where $V(H) \subseteq V(G)$ maps each $e = (u,v) \in E(H)$ to a (u,v)-path $\Pi_{H\to G}(e)$ in G. The *congestion* of $\Pi_{H\to G}$ is

$$\text{cong}(\Pi_{H\to G}) \coloneqq \max_{e_G\in E(G)} \frac{\sum_{e_H\in E(H):e_G\in\Pi_{H\to G}(e_H)} \boldsymbol{c}_H(e_H)}{\boldsymbol{c}_G(e_G)}.$$

Cut-Matching Game. The cut-matching game is a framework for constructing expanders from the interaction of two players: the cut player and the matching player. Suppose we want to construct an expander over vertices V starting from an initially empty graph. The game proceeds in rounds, and in each round the cut player first computes a bisection (A,B) of V, and then the matching player returns a (perfect) matching from A to B, which is then added to the graph. The goal of the cut player is to compute the bisections in such a way that after a small number of rounds, the resulting graph becomes an expander regardless of what perfect matchings the matching player returns. [Lou10] extended the randomized cut player for undirected graphs and its analysis from [KRV06] to work in directed graphs. This can be straightforwardly generalized to the capacitated case. For

vertex weights $\boldsymbol{\nu}_A$ and $\boldsymbol{\nu}_B$ with $\|\boldsymbol{\nu}_A\|_1 \leq \|\boldsymbol{\nu}_B\|_1$, a $(\boldsymbol{\nu}_A, \boldsymbol{\nu}_B)$-*perfect (capacitated) matching* is an $(M, \boldsymbol{c}_M)$ such that $\deg^+_{M,\boldsymbol{c}_M}(v) = \boldsymbol{\nu}_A(v)$ and $\deg^-_{M,\boldsymbol{c}_M}(v) \leq \boldsymbol{\nu}_B(v)$ for all $v \in V$.

Theorem A.3.6 ([KRV06; Lou10])**.** *Given n vertices V and a vector $\boldsymbol{\nu} \in \mathbb{N}^V$ with entries bounded by U, there is a randomized algorithm that computes in sequence $t_{CMG} = O(\log^2(nU))$ vector pairs $(\boldsymbol{\nu}_A^{(i)}, \boldsymbol{\nu}_B^{(i)})$ with $\boldsymbol{\nu}_A^{(i)} + \boldsymbol{\nu}_B^{(i)} \leq \boldsymbol{\nu}$ and $\|\boldsymbol{\nu}_A^{(i)}\|_1 \leq \|\boldsymbol{\nu}_B^{(i)}\|_1$ such that if it is given $(\boldsymbol{\nu}_A^{(i)}, \boldsymbol{\nu}_B^{(i)})$-perfect capacitated matching $(M_i, \boldsymbol{c}_i)$ after it outputs each $(\boldsymbol{\nu}_A^{(i)}, \boldsymbol{\nu}_B^{(i)})$, then in the end it outputs a ψ_{CMG}-expander $(W, \boldsymbol{c}_W)$ with edges $M_1 \cup \cdots \cup M_{t_{CMG}}$ such that $\boldsymbol{\nu}(v) \leq \deg_{W,\boldsymbol{c}_W}(v) \leq t_{CMG} \cdot \boldsymbol{\nu}(v)$ for all $v \in V$, where $\psi_{CMG} = \Omega\left(\frac{1}{\log^2(nU)}\right)$. The algorithm runs in $\widetilde{O}(n + |M_1| + \cdots + |M_{t_{CMG}}|)$ time.*

A.4 Push-Relabel Algorithm

Suppose that $G = (V, E)$ is a directed graph, which we want to solve the maximum flow problem on. In this section, we will also assume that we are given a *weight function* $\boldsymbol{w} \in \mathbb{N}^E$ on the edges as additional input. This weight function will serve as a "hint" and will help us to find a good approximate flow more efficiently. In an ideal world, we would want the weight function to satisfy the following properties:

- There is some "short" flow $\boldsymbol{f}$ which is a good approximation to the optimal maximum flow. With "short", we mean that the average $\boldsymbol{w}$-length $\frac{\boldsymbol{w}(\boldsymbol{f})}{|\boldsymbol{f}|}$ is something like $\tilde{O}(n)$.
- The sum $\sum_{e \in E} \frac{1}{\boldsymbol{w}(e)}$ is "small", something like $\tilde{O}(n)$.

The goal of this section is to design a version of the *push-relabel*[14] algorithm, originally due to [GT88], that, when the above properties are fulfilled, will find a constant-approximation to the maximum flow efficiently. Hence, given the following Theorem A.4.1, solving the maximum flow problem in $n^{2+o(1)}$ time reduces to efficiently finding a "good" weight function $\boldsymbol{w}$.

Theorem A.4.1 (Push-Relabel)**.** *Suppose we have a maximum flow instance $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ consisting of an n-vertex m-edge directed graph $G = (V, E)$, edge capacities $\boldsymbol{c} \in \mathbb{N}^E$, and integral source and sink vectors $\boldsymbol{\Delta}, \boldsymbol{\nabla} \in \mathbb{N}^V$. Additionally, suppose we have a weight function $\boldsymbol{w} \in \mathbb{N}^E_{>0}$ and height parameter $h \in \mathbb{N}$. Then there is an algorithm—Algorithm A.1:* PushRelabel$(G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla}, \boldsymbol{w}, h)$*—that in $\tilde{O}\left(m + n + \sum_{e \in E} \frac{h}{\boldsymbol{w}(e)}\right)$ time finds a feasible integral flow $\boldsymbol{f}$ such that*

[14] Also sometimes called *preflow-push*, although our version will maintain proper flows and not preflows.

(i) the $\boldsymbol{w}$-distance in the residual graph $G_{\boldsymbol{f}}$ between any unsaturated source s ($\boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0$) and any unsaturated sink t ($\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$) is at least $\operatorname{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}}}(s,t) > 3h$,

(ii) the average $\boldsymbol{w}$-length of the flow is $\frac{\boldsymbol{w}(\boldsymbol{f})}{|\boldsymbol{f}|} \leq 9h$, and

(iii) $\boldsymbol{f}$ is a $\frac{1}{6}$-approximation of $\boldsymbol{f}^*_{\boldsymbol{w},h}$—the optimal (not necessarily integral) flow with average $\boldsymbol{w}$-length $\frac{\boldsymbol{w}(\boldsymbol{f}^\star_{\boldsymbol{w},h})}{|\boldsymbol{f}^\star_{\boldsymbol{w},h}|} \leq h$.

A.4.1 Push-Relabel Finds an Approximate Short Flow

Before proving Theorem A.4.1 fully, we show how (iii) is implied by (i) and (ii).

Lemma A.4.2. *Let $\boldsymbol{f}^\star$ be a (possibly fractional) feasible $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow where $\boldsymbol{w}(\boldsymbol{f}^\star) \leq |\boldsymbol{f}^\star| \cdot h$, and let $\boldsymbol{f}$ be a (possibly fractional) feasible $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow which satisfies (i) and (ii) of Theorem A.4.1; then $|\boldsymbol{f}| \geq \frac{1}{6}|\boldsymbol{f}^\star|$.*

Proof. We extend the graph G to G' by adding a super-source s and super-sink t, and adding edges (s,v) and (v,t) with capacities $\boldsymbol{c}(s,v) = \boldsymbol{\Delta}(v)$ respectively $\boldsymbol{c}(v,t) = \boldsymbol{\nabla}(v)$ and weights $\boldsymbol{w}(s,v) = \boldsymbol{w}(v,t) = 0$. This lets us now consider the (s,t)-flow problem with $\boldsymbol{\Delta}' = \infty \cdot \mathbf{1}_s$ and $\boldsymbol{\nabla}' = \infty \cdot \mathbf{1}_t$. Similarly, the flow $\boldsymbol{f}$ and $\boldsymbol{f}^\star$ can be extended to the graph G', by setting $\boldsymbol{f}(s,v) = \boldsymbol{\Delta}(v) - \mathbf{ex}_{\boldsymbol{f}}(v)$ and $\boldsymbol{f}(v,t) = \mathbf{abs}_{\boldsymbol{f}}(v)$ and similarly for $\boldsymbol{f}^\star$.

Assume for contradiction that $|\boldsymbol{f}| < \frac{1}{6}|\boldsymbol{f}^\star|$. Consider the flow $\boldsymbol{f}'$ in the residual graph $G'_{\boldsymbol{f}}$ where we first send $\boldsymbol{f}$ backward, making the residual graph equal to G', and then send $\boldsymbol{f}^\star$ forwards (i.e., $\boldsymbol{f}' = \boldsymbol{f}^\star - \boldsymbol{f}$). We note that $\boldsymbol{f}'$ is a feasible flow in $G'_{\boldsymbol{f}}$, since $\mathbf{ex}_{\boldsymbol{f}'}(s) = \infty$ and $\mathbf{ex}_{\boldsymbol{f}'}(v) = 0$ for all $v \neq s$, so we have $\mathbf{0} \leq \mathbf{ex}_{\boldsymbol{f}'} \leq \boldsymbol{\Delta}'_{\boldsymbol{f}} = \infty \cdot \mathbf{1}_s$.[15] We know that $|\boldsymbol{f}'| = |\boldsymbol{f}^\star| - |\boldsymbol{f}| > \frac{5}{6}\boldsymbol{f}^\star$. We also know that, by definition, $\boldsymbol{w}(\boldsymbol{f}') \leq \boldsymbol{w}(\boldsymbol{f}^\star) + \boldsymbol{w}(\boldsymbol{f})$ with $\boldsymbol{w}(\boldsymbol{f}) \leq 9h|\boldsymbol{f}|$, by (ii). This gives

$$\frac{\boldsymbol{w}(\boldsymbol{f}')}{|\boldsymbol{f}'|} \leq \frac{|\boldsymbol{f}^\star| \cdot h + |\boldsymbol{f}| \cdot 9h}{\frac{5}{6}|\boldsymbol{f}^\star|} < \frac{|\boldsymbol{f}^\star| \cdot h + |\boldsymbol{f}^\star| \cdot \frac{3}{2}h}{\frac{5}{6}|\boldsymbol{f}^\star|} = 3h$$

meaning that, by an averaging argument, there must exist an (s,t)-path in the residual graph $G'_{\boldsymbol{f}}$ (and thus also a path in $G_{\boldsymbol{f}}$ between an unsaturated source and unsaturated sink) of length less than $3h$. However, this contradicts (i). □

Remark A.4.3. It is worth noting that the reference flow $\boldsymbol{f}^\star$ in Lemma A.4.2 needs not be integral. Nevertheless, this does not contradict the large integrality gap or the hardness-of-approximation results of the "bounded-length flow polytope" (see, e.g., [GKRSY03; BEHKKPSS10]), since the flow $\boldsymbol{f}$ we find will have length slightly larger than h (i.e., the $9h$ term in Theorem A.4.1(ii)).

[15] In the original graph G, the flow $\boldsymbol{f}'$ would not necessarily be feasible in $G_{\boldsymbol{f}}$ as $\mathbf{ex}_{\boldsymbol{f}^\star}(v)$ could be less than $\mathbf{ex}_{\boldsymbol{f}}(v)$ for some vertex v. This is the reason why we work in G' instead.

An immediate corollary of the Lemma A.4.2 is that the existence of short, possibly fractional flow implies the existence of short integral flow.

Corollary A.4.4. *If there is a (possibly fractional) feasible* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$*-flow* $\boldsymbol{f}^\star$ *where* $\boldsymbol{w}(\boldsymbol{f}^\star) \leq |\boldsymbol{f}^\star| \cdot h$*, then there is an integral feasible* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$*-flow* $\boldsymbol{f}$ *with* $|\boldsymbol{f}| \geq \frac{1}{6}|\boldsymbol{f}^\star|$ *with* $\boldsymbol{w}(\boldsymbol{f}) \leq |\boldsymbol{f}| \cdot O(h)$*.*

Proof. Let $\boldsymbol{f}$ be the integral flow obtained by repeatedly finding augmenting paths P of weight $\boldsymbol{w}(P) \leq 3h$ until no such paths exist. The flow $\boldsymbol{f}$ clearly satisfies Theorem A.4.1(i) and (ii). The corollary follows from Lemma A.4.2. □

A.4.2 Implementation

We now present the pseudocode in Algorithm A.1. Our implementation differs from a textbook push-relabel algorithm in the following ways:

- We restrict our algorithm to $9h$ levels; vertices v with level $\boldsymbol{\ell}(v) > 9h$ are marked as *dead*.
- Our algorithm allows for edge-length $\boldsymbol{w}(e)$ for each edge. While a textbook push-relabel algorithm can send flow on *admissible* edges (u, v) where the level $\boldsymbol{\ell}(u) = \boldsymbol{\ell}(v) + 1$, we instead call an edge *admissible* when $\boldsymbol{\ell}(u) \approx \boldsymbol{\ell}(v) + \boldsymbol{w}(e)$. This is useful to obtain the faster running time, since, as we will see, an edge e only changes between being admissible/inadmissible $O\left(\frac{h}{\boldsymbol{w}(e)}\right)$ times.
- Our algorithm employs an aggressive relabeling rule: as long as some vertex (which has no unsaturated sink capacity) does not have any *admissible* outgoing edge, we relabel it (even if it does not have any excess flow).
- The above point means that whenever we find an augmenting flow path, we can push a unit of flow all the way from a source to a sink directly, and that the length of this flow path is only $O(h)$ (allowing us to argue (ii) and hence also (iii)). Another consequence is that the flow $\boldsymbol{f}$ maintained by the algorithm will always be a proper flow, and not a *preflow*, as is usual in push-relabel implementations.

Remark A.4.5. Even on a directed path of length n, our push-relabel algorithm would require $\Omega(n^2)$ time. This is unlike most variants of push-relabel that usually prioritize pushing instead of relabeling, which would take $O(n)$ time on a path. While our relabel-prioritized variant can compute an approximate maximum flow with small average length, which is crucial for us, it also becomes our bottleneck. This raises the exciting question of whether a weighted version of the push-prioritized push-relabel algorithm can be devised so that, given a good weight function (or something similar), it runs in $m^{1+o(1)}$ time and computes a $(1/n^{o(1)})$-approximate maximum flow. In a graph with unit vertex capacities, a weight function induced by the DAG of the maximum flow will guide a push-prioritized algorithm to run

in linear time, but as of now it is unclear how to identify such a "good" weight function without first computing the maximum flow.

Recall that the push relabel algorithm runs in the residual graph $G_{\boldsymbol{f}} = (V, \overrightarrow{E} \cup \overleftarrow{E})$ which is defined as follows: for each edge $e = (u, v) \in E$, we have a forward edge $\overrightarrow{e} = (u, v) \in \overrightarrow{E}$ and a backward edge $\overleftarrow{e} = (v, u) \in \overleftarrow{E}$ with residual capacities $\boldsymbol{c}_{\boldsymbol{f}}(\overrightarrow{e}) = \boldsymbol{c}(e) - \boldsymbol{f}(e)$ and $\boldsymbol{c}_{\boldsymbol{f}}(\overleftarrow{e}) = \boldsymbol{f}(e)$. We will often use e and $\overrightarrow{e}$ interchangeably (e.g., the flow $\boldsymbol{f}$ will be defined on $\overrightarrow{E}$), and often when referring to $G_{\boldsymbol{f}}$ as a graph we will ignore all edges with residual capacity 0 (e.g., when talking about distances in $G_{\boldsymbol{f}}$).

A.4.3 Proof of the Push Relabel Algorithm

We begin by showing some helpful invariants.

Lemma A.4.6. *Throughout the run of Algorithm A.1, the following invariants hold:*

(I-1) $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) < 3\boldsymbol{w}(e)$, *for all* $e = (u, v) \in \overrightarrow{E} \cup \overleftarrow{E}$ *with* $\boldsymbol{c}_{\boldsymbol{f}}(e) > 0$.

(I-2) $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) > \boldsymbol{w}(e)$, *for all* $e = (u, v) \in \overrightarrow{E} \cup \overleftarrow{E}$ *marked* admissible,

(I-3) $\boldsymbol{\ell}(v) \leq 9h$ *for each* alive *vertex* v, $\boldsymbol{\ell}(v) > 9h$ *for each* dead *vertex* v, *and* $\boldsymbol{\ell}(t) = 0$ *for all unsaturated sinks* t ($\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$).

Proof. It is easy to verify that all invariants hold initially.

We begin with the invariants (I-1) and (I-2). Consider some edge $e = (u, v)$, and let $\boldsymbol{\ell}^{\text{old}}(u), \boldsymbol{\ell}^{\text{old}}(v)$ be the levels of u and v the last time edge e was marked as admissible or inadmissible. Note that $\boldsymbol{\ell}(u) \in [\boldsymbol{\ell}^{\text{old}}(u), \boldsymbol{\ell}^{\text{old}}(u) + \boldsymbol{w}(e) - 1]$ and $\boldsymbol{\ell}(v) \in [\boldsymbol{\ell}^{\text{old}}(v), \boldsymbol{\ell}^{\text{old}}(v) + \boldsymbol{w}(e) - 1]$, as if the levels of u (or v) had increased by at least $\boldsymbol{w}(e)$, then there must have been a point where $\boldsymbol{w}(e)$ divided $\boldsymbol{\ell}(u)$ (or $\boldsymbol{\ell}(v)$).

1. If e was marked as inadmissible and $\boldsymbol{c}_{\boldsymbol{f}}(e) > 0$, we know $\boldsymbol{\ell}^{\text{old}}(u) - \boldsymbol{\ell}^{\text{old}}(v) < 2\boldsymbol{w}(e)$, and hence that $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) < 2\boldsymbol{w}(e) + (\boldsymbol{w}(e) - 1)$.

2. If e was marked as admissible, we know $\boldsymbol{\ell}^{\text{old}}(u) - \boldsymbol{\ell}^{\text{old}}(v) \geq 2\boldsymbol{w}(e)$, and hence that $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) \geq 2\boldsymbol{w}(e) - (\boldsymbol{w}(e) - 1)$. Additionally, we note that as long as e is admissible, the quantity $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v)$ cannot increase (since it only increases when $\boldsymbol{\ell}(u)$ goes up, which only happens if we relabel u, which in turn only happens when there is no admissible outgoing edge of u). Because at the last point when e was inadmissible we had $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) < 3\boldsymbol{w}(e) - 1$, we know that $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) < 3\boldsymbol{w}(e)$ now too.

Invariant (I-3) is easy to see, since any vertex of level $> 9h$ is marked dead from the graph, and the unsaturated sinks (i.e., those t with $\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$) are never relabeled. □

Algorithm A.1: PUSHRELABEL$(G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla}, \boldsymbol{w}, h)$

Initialize $\boldsymbol{f}$ as the empty flow.
Let $\boldsymbol{\ell}(v) = 0$ for all $v \in V$. `// levels`
Mark each edge $e \in \overrightarrow{E} \cup \overleftarrow{E}$ as *inadmissible* and all vertices as *alive*.
function RELABEL(v)
    Set $\boldsymbol{\ell}(v) \leftarrow \boldsymbol{\ell}(v) + 1$.
    if $\boldsymbol{\ell}(v) > 9h$ **then**
        mark v as *dead* and **return**.
    for *each edge* $e \ni v$ *where* $\boldsymbol{w}(e)$ *divides* $\boldsymbol{\ell}(v)$ **do**
        Let $(x, y) = e$.
        if $\boldsymbol{\ell}(x) - \boldsymbol{\ell}(y) \geq 2\boldsymbol{w}(e)$ *and* $\boldsymbol{c_f}(e) > 0$ **then** mark e as *admissible*.
        else mark e as *inadmissible*.
main loop
    while *there is an alive vertex* v *with* $\boldsymbol{\nabla_f}(v) = 0$ *and without an admissible out-edge* **do**
        RELABEL(v)
    if *there is some alive vertex* s *with* $\boldsymbol{\Delta_f}(s) > 0$ **then**
        `//` P `is an "augmenting path"`
        Trace a path P from s to some sink t, by arbitrarily following admissible out-edges.
        Let $c^{\text{augment}} \leftarrow \min\{\boldsymbol{\Delta_f}(s), \boldsymbol{\nabla_f}(t), \min_{e \in P} \boldsymbol{c_f}(e)\}$.
        for $e \in P$ **do** `// Augment` $\boldsymbol{f}$ `along` P
            if e *is a forward edge* **then** $\boldsymbol{f}(e) \leftarrow \boldsymbol{f}(e) + c^{\text{augment}}$.
            else $\boldsymbol{f}(e') \leftarrow \boldsymbol{f}(e') - c^{\text{augment}}$, where e' is the corresponding forward edge to e.
            Adjust residual capacities $\boldsymbol{c_f}$ of e and the corresponding reverse edge.
            if $\boldsymbol{c_f}(e) = 0$ **then** mark e as *inadmissible*.
        `//` $\boldsymbol{\Delta_f}(s)$ `and` $\boldsymbol{\nabla_f}(t)$ `goes down by` c^{augment}
    else return $\boldsymbol{f}$

Because the algorithm relabels all alive vertices (except unsaturated sources) until they have an admissible outgoing edge, we note that when the algorithm tries to trace a path P by arbitrarily following admissible edges, the path P must eventually end in an unsaturates sink. Indeed, at this time, all alive vertices which does not have admissible outgoing edges are exactly the unsaturated sinks. Moreover, traversing an admissible edge, by (I-2), decreases the level $\boldsymbol{\ell}$, so this process cannot go on forever and P must eventually end in an unsaturated sink.

We will also need a bound on the number of augmentations the algorithm performs:

Lemma A.4.7. *Every edge e (or its reverse) is only saturated (i.e., has $\boldsymbol{c_f}(e) = c^{\text{augment}}$) in at most $O(\frac{h}{\boldsymbol{w}(e)})$ augmenting paths. Thus, there are at most $O(n + \sum_{e\in E} \frac{h}{\boldsymbol{w}(e)})$ many augmenting paths found by the algorithm.*

Proof. Indeed, whenever edge $e = (u, v)$ is fully saturated as part of an augmenting path, it will be marked as inadmissible. At this point we have $\boldsymbol{\ell}(u) - \boldsymbol{\ell}(v) > \boldsymbol{w}(e)$ by (I-2), and it (or rather, its reverse $e' = (v, u)$) will only ever be marked as admissible when $\boldsymbol{\ell}(v) - \boldsymbol{\ell}(u) \geq 2\boldsymbol{w}(e)$. This means that the sum $\boldsymbol{\ell}(u) + \boldsymbol{\ell}(v)$ must have increased (since levels only ever increase) by $\Theta(\boldsymbol{w}(e))$. Since $\boldsymbol{\ell}(u) + \boldsymbol{\ell}(v) \leq 18h$, this can only happen $O(\frac{h}{\boldsymbol{w}(e)})$ times.

For the second part of the lemma, we note that for each augmenting path, either an edge is saturated, or a source/sink vertex gets saturated. The former can happen at most $O(\sum \frac{h}{\boldsymbol{w}(e)})$ times for each edge e, and the latter can happen at most once for each vertex. This bounds the number of augmenting paths. □

We now resume to prove the guarantees listed in Theorem A.4.1 (recall that (iii) was already shown in Lemma A.4.2).

Returns Short Flows. We begin by showing that Algorithm A.1 returns a flow $\boldsymbol{f}$ such that $\boldsymbol{w}(\boldsymbol{f}) \leq 9h \cdot |\boldsymbol{f}|$ (thus proving (ii)). Indeed, each augmenting path $P = (s = v_0, v_1, v_2, \ldots, v_k = t)$ our algorithm finds consists of admissible edges. This means that $\boldsymbol{w}(P) = \sum_{e\in P} \boldsymbol{w}(e) \leq \sum_{i=0}^{k-1} \boldsymbol{\ell}(v_i) - \boldsymbol{\ell}(v_{i+1}) = \boldsymbol{\ell}(s) - \boldsymbol{\ell}(t) \leq 9h$ (by (I-2) and (I-3)). Hence $\boldsymbol{w}(P) \leq 9h$. Now, by summing over all augmenting paths, observe that $\boldsymbol{w}(\boldsymbol{f}) \leq \sum_P \boldsymbol{w}(P) \cdot |\boldsymbol{f}_P|$ because the flow paths may only cancel in the final flow $\boldsymbol{f}$. Therefore, we have $\boldsymbol{w}(\boldsymbol{f}) \leq \sum_P 9h \cdot |\boldsymbol{f}_P| = 9h \cdot |\boldsymbol{f}|$.

Source-to-Sink Distance is Large in Residual Graph. We now argue that the shortest source-to-sink path in the residual graph $G_{\boldsymbol{f}}$ must have $\boldsymbol{w}$-length more than $3h$ (thus proving (i)). Consider any (s, t)-path $P = (s = v_0, v_1, v_2, \ldots, v_k = t)$ in the residual graph, where s is an unsaturated source and t an unsaturated sink. Then $3\boldsymbol{w}(P) = \sum_{e\in P} 3\boldsymbol{w}(e) > \sum_{i=0}^{k-1} \boldsymbol{\ell}(v_i) - \boldsymbol{\ell}(v_{i+1}) = \boldsymbol{\ell}(s) - \boldsymbol{\ell}(t) > 9h$. The first inequality is by (I-1). The last inequality is because every unsaturated source s has level $\boldsymbol{\ell}(s) > 9h$ at termination (indeed, s must be *dead*, as otherwise the algorithm would not have terminated), and every unsaturated sink t always has level $\boldsymbol{\ell}(t) = 0$ by (I-3). Hence $\boldsymbol{w}(P) > 3h$, which is what we wanted.

Running Time. We now argue that we can implement Algorithm A.1 in running time bounded by $\tilde{O}\left(m + n + \sum_{e\in E} \frac{h}{\boldsymbol{w}(e)}\right)$. Here, we only argue that this is the case if the graph is unit-capacitated, i.e., when $\boldsymbol{c}(e) = 1$ for all $e \in E$. The full argument on the running time bound in capacitated graphs requires keeping track

of the admissible edges using dynamic trees (e.g., link-cut trees [ST83]) to speed up parts of the algorithm—the discussion of which we postpone to Section A.8. We proceed by analyzing the running time of the different parts of the algorithm.

- The initialization steps takes $O(n+m)$ time.
- For each vertex v we can keep track of its admissible out-edges in a linked list (to support addition and removal in constant time). Additionally, we can keep track of a list of all vertices v which have no admissible out-edges (so that we can find such a vertex efficiently in constant time).
- Relabel:
  - For each vertex v, the relabel operation is run at most $O(h)$ times. This would give an extra factor of $O(nh)$.[16] To avoid this, when we perform a relabel operation, we can increase the level of a vertex by more than one. Note that a vertex will only get a new admissible out-edge when some incident edge e has $\boldsymbol{w}(e)$ which divides the new level $\boldsymbol{\ell}(v)$. Thus, for vertex v, we can immediately raise the level to the next multiple of $\boldsymbol{w}(e)$ for any of the adjacent edges. In total, vertex v will thus only visit at most $O\left(\sum_{e\in\delta(v)} \frac{h}{\boldsymbol{w}(e)}\right)$ levels. In total, over all vertices, we will thus have at most $O\left(\sum_{e\in E} \frac{h}{\boldsymbol{w}(e)}\right)$ relabel operations.
  - We also argue that the for-loop to mark edges as (in)admissible is efficient. Consider an edge $e=(u,v)$. It will be considered $O\left(\frac{h}{\boldsymbol{w}(e)}\right)$ many times in the for-loop (since at most this many times, $\boldsymbol{w}(e)$ will divide $\boldsymbol{\ell}(v)$ or $\boldsymbol{\ell}(u)$). In total, this for-loop thus accounts for $O\left(\sum_{e\in E} \frac{h}{\boldsymbol{w}(e)}\right)$ running time. Indeed, we can quickly identify which edges to loop over by storing, for each vertex, a dictionary, where entry k maps to all edges whose weights divide k; such a dictionary can be populated as an initialization step.
- Processing Augmenting Paths:
  - By Lemma A.4.7, there are only $O\left(n+\sum_{e\in E} \frac{h}{\boldsymbol{w}(e)}\right)$ augmenting paths found. Using dynamic trees (as is a standard speed-up for push-relabel algorithms), we can in fact support each augmentation in $O(\log n)$ time by keeping track of trees where vertex v has an arbitrary admissible out-edge as parent, and using a dynamic tree data structure to support "add" and "find-min" operations on a vertex-to-root path (note: the roots will exactly be the unsaturated sinks). However, we postpone this discussion to Section A.8.

[16] For our purposes this term is actually acceptable.

– Here we instead argue the bound in the case of unit-capacitated graphs: The amount of work we do when we find an augmenting path is proportional to the length of this augmenting path. Thus we charge one unit of work to each edge e_i on the path. Since the graph is of unit-capacity, *all* edges e_i on the path will be saturated. Hence, by Lemma A.4.7, we know that the edge e_i will only be charged cost $O\left(\frac{h}{\boldsymbol{w}(e)}\right)$ throughout the run of the algorithm, for a total cost of $O\left(\sum_{e\in E}\frac{h}{\boldsymbol{w}(e)}\right)$.

The above discussion concludes the proof of Theorem A.4.1.

Additional Property of Finding Almost Shortest Paths. The push relabel algorithm works by finding augmenting paths one by one. Say the paths, in order, are $P_1, P_2, \ldots, P_{|\boldsymbol{f}|}$. Let $\boldsymbol{f}_i$ be the flow induced by paths $P_1, \ldots, P_i$, and note that the path P_{i+1} is a path in the residual graph $G_{\boldsymbol{f}_i}$. When bootstrapping our algorithm to find an expander decomposition, we will later need the additional property that the augmenting paths our algorithm finds cannot be "shortcutted" significantly. We prove the following lemma.

Lemma A.4.8. *Consider the state of the algorithm just before the i-th path P_i is augmented along. Then, for any vertices s and t we have $\boldsymbol{\ell}(s)-\boldsymbol{\ell}(t) \le 3\mathrm{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}_{i-1}}}(s,t)$. In particular, this means that any subpath P' of P_i, between vertices s and t, has weight at most $\boldsymbol{w}(P') \le 3\mathrm{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}_{i-1}}}(s,t)$.*

Proof. Consider the shortest (s,t)-path $Q = (v_1, v_2, \ldots, v_{|Q|})$ (with $v_1 = s$ and $v_{|Q|} = t$) in the residual graph $G_{\boldsymbol{f}_{i-1}}$. Then we have

$$3\boldsymbol{w}(Q) = \sum_{e\in Q} 3\boldsymbol{w}(e) \ge \sum_{i=1}^{|Q|-1} (\boldsymbol{\ell}(v_i) - \boldsymbol{\ell}(v_{i+1})) = \boldsymbol{\ell}(s) - \boldsymbol{\ell}(t)$$

by (I-1). This proves the first part of the lemma.

The second part is similar, but now using (I-2) and the fact that all edges on the path P' are admissible. Suppose $P' = (u_1, \ldots, u_{|P'|})$ (with $u_1 = s$ and $u_{|P'|} = t$). Then we have

$$\boldsymbol{w}(P') = \sum_{e\in P'} \boldsymbol{w}(e) \le \sum_{i=1}^{|P'|-1} (\boldsymbol{\ell}(u_i) - \boldsymbol{\ell}(u_{i+1})) = \boldsymbol{\ell}(s) - \boldsymbol{\ell}(t) \le 3\mathrm{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}_{i-1}}}(s,t). \quad \square$$

A.4.4 Application: Approximate Max-Flow in DAGs

While it is non-trivial to find a "good" weight function for arbitrary graphs[17], the special case of directed acyclic graphs (DAGs) turns out to be easy. Therefore we

[17] Finding a "good" weight function efficiently is exactly what we do in the remainder of our paper, by the use of expander decomposition.

can immediately obtain (by using our push-relabel algorithm Algorithm A.1) a relatively simple, combinatorial, linear-time-for-dense-graphs, constant-approximation algorithm for maximum flow in DAGs.

Corollary A.4.9. *One can find a $\Theta(1)$-approximate maximum flow in a DAG in $\tilde{O}(n^2)$ time.*

Proof. Let $\boldsymbol{\tau} \in [n]^V$ be the topological order of the vertices (which can be found in $O(n+m)$ time): that is $\boldsymbol{\tau}_u < \boldsymbol{\tau}_v$ for each edge $e = (u, v)$. Then let $\boldsymbol{w}(e) = |\boldsymbol{\tau}_v - \boldsymbol{\tau}_u|$. Indeed, any flow path in the maximum flow $\boldsymbol{f}^\star$ will have $\boldsymbol{w}$-length at most n. Thus, we can invoke Algorithm A.1 and Theorem A.4.1 with $h = n$, getting a $\frac{1}{6}$-approximation of the maximum flow. □

A.5 Solving Maximum Flow

In this section, we show how to compute maximum flow exactly using the weighted push-relabel algorithm from Section A.4 given an expander hierarchy defined below.

Definition A.5.1. Given a capacitated graph $(G, \boldsymbol{c})$, a partition $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of $E(G)$ is a *ϕ-expander hierarchy* of $(G, \boldsymbol{c})$ with *height* $\eta(\mathcal{H}) = \eta$ if

1. D is acyclic,
2. each $e \in X_i$ is contained in some strongly connected component of $G \setminus X_{>i}$, and
3. X_i is a ϕ-expanding in $(G \setminus X_{>i}, \boldsymbol{c})$.[18]

When the graph is of unit-capacity (i.e., when $\boldsymbol{c} = \mathbf{1}$), we will leave $\boldsymbol{c}$ out from the notation. We remark that our algorithm for constructing an expander hierarchy actually guarantees that X_i is a *separator* in $G \setminus X_{>i}$. See Section A.7 for more details. As we will see, for our purposes, we should think of $\phi = n^{o(1)}$ and $\eta = O(\log n)$.

Consider a ϕ-expander hierarchy $\mathcal{H}$ of $(G, \boldsymbol{c})$. An edge $e \in D$ is a *DAG edge*. An edge $e \in X_i$ for some i is an *expanding edge*, and more specifically a *level-i* expanding edge. Let $G_i := G \setminus X_{>i}$, in which each $C \in \mathrm{SCC}(G_i)$ is a *level-i expander*. Note that, by definition, the level-i expanders $\mathrm{SCC}(G_i)$ form a refinement of the level-$(i+1)$ expanders $\mathrm{SCC}(G_{i+1})$. See Figure A.1 for illustration.

Note that Definition A.5.1 differs from the undirected expander hierarchy of [GRST21] in that we are not contracting strongly connected components as we go up in the hierarchy. Indeed, it is impossible to ensure the boundary-linkedness of all inter-cluster edges as in [GRST21] due to the presence of DAG edges in directed graphs. We also remark that our ϕ-expander hierarchy precisely generalizes the expander hierarchy notion from [PT07, Definition 2] in undirected graphs to directed graphs. Note that in undirected graphs, we would always have $D = \emptyset$.

[18] Recall that this means that $X_i \cap C$ is ϕ-expanding in $(C, \boldsymbol{c})$, for each strongly connected component C of $G \setminus X_{>i}$.

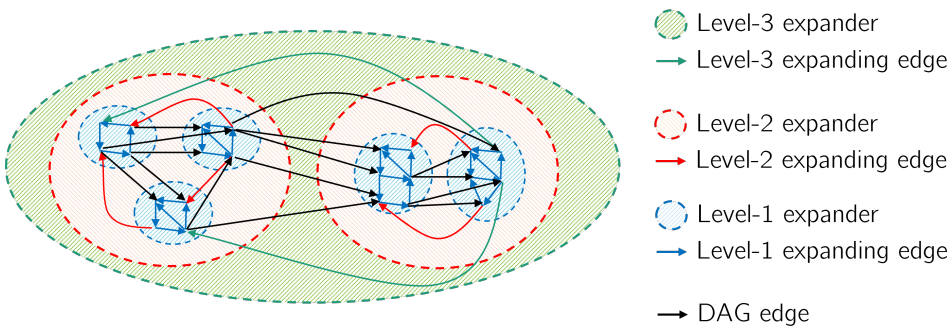


Figure A.1: An example of an expander hierarchy with 3 levels

A.5.1 Weight Function

As alluded to in previous sections, computing maximum flow boils down to the design of a good weight function which we can run the push relabel algorithm from Section A.4 on. Recall that we want our weight function $\boldsymbol{w}$ to satisfy (1) that some approximate maximum flow is short with respect to $\boldsymbol{w}$, and (2) that the sum of inverses $\sum_{e\in E}\frac{1}{\boldsymbol{w}(e)}$ is small (for an efficient running time). We prove (1) in Theorem A.5.6 and (2) in Claim A.5.5.

Let $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ be a ϕ-expander hierarchy of $(G, \boldsymbol{c})$ (we show how to compute $\mathcal{H}$ in Section A.7). We now define the weight function $\boldsymbol{w}_{\mathcal{H}} \in \mathbb{N}^E_{>0}$ of G induced by the hierarchy $\mathcal{H}$. We will show in the remainder of the section that it indeed satisfies the desired properties. The choice of our weight function is inspired by the topological order[19] of DAGs, and we consider the following notion of respecting topological order.

Definition A.5.2. A topological order $\boldsymbol{\tau} \in \mathbb{N}^V$ of D is *$\mathcal{H}$-respecting* if for each level i and each $C \in \mathrm{SCC}(G_i)$ the set $\boldsymbol{\tau}(C) \coloneqq \{\boldsymbol{\tau}_v : v \in C\}$ is contiguous In other words, it contains precisely the set of numbers between

$$\boldsymbol{\tau}_{\min}(C) \coloneqq \min \boldsymbol{\tau}(C) \quad \text{and} \quad \boldsymbol{\tau}_{\max}(C) \coloneqq \max \boldsymbol{\tau}(C).$$

Note that given a hierarchy $\mathcal{H}$, an $\mathcal{H}$-respecting $\boldsymbol{\tau}$ can be easily computed in $O(m\eta)$ time by the following lemma whose proof is deferred to Section A.10.

Lemma A.5.3. *Given an expander hierarchy $\mathcal{H}$, in $O(m\eta)$ time we can compute an $\mathcal{H}$-respecting topological order $\boldsymbol{\tau}$.*

As a result, in the remainder of this paper whenever there is a hierarchy $\mathcal{H}$ we assume we also have a corresponding $\mathcal{H}$-respecting topological order $\boldsymbol{\tau}$ (which might not be unique). The weight function $\boldsymbol{w}_{\mathcal{H}}$ induced by $\mathcal{H}$ (and $\boldsymbol{\tau}$) is simply defined as

$$\boldsymbol{w}_{\mathcal{H}}(e) \coloneqq |\boldsymbol{\tau}_v - \boldsymbol{\tau}_u|. \tag{A.1}$$

[19] Recall that a *topological order* of an acyclic D is a permutation $\boldsymbol{\tau} \in \mathbb{N}^V$ such that $\boldsymbol{\tau}_u < \boldsymbol{\tau}_v$ for every $(u, v) \in E(D)$. A topological order always exists and can be computed in $O(m)$ time [Tar72].

Observation A.5.4. *A level-i expanding edge $e = (u, v)$ has $\boldsymbol{w}_{\mathcal{H}}(e) \leq |C|$, where C is the level-i expander in which e is expanding.*

Recall that the running time of our push-relabel algorithm depends on the sum of the inverses of the edge weights, which we claim below is small.

Claim A.5.5. *For a simple graph G it holds that $\sum_{e \in E} \frac{1}{\boldsymbol{w}_{\mathcal{H}}(e)} = O(n \log n)$.*

Proof. Since the graph is simple, there is at most two edges (one in each direction) between a pair of vertices $\{u, v\}$. Hence,

$$\sum_{e \in E} \frac{1}{\boldsymbol{w}_{\mathcal{H}}(e)} \leq \sum_{\substack{\boldsymbol{\tau}_u, \boldsymbol{\tau}_v \in [n] \\ \boldsymbol{\tau}_u \neq \boldsymbol{\tau}_v}} \frac{1}{|\boldsymbol{\tau}_u - \boldsymbol{\tau}_v|} = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j - i} = O(n \log n). \qquad \square$$

The Maximum Flow Algorithm. The key structural lemma that our max-flow algorithm relies on is that it is without loss of optimality for us to focus only on flow paths that are relatively short with respect to the weight function $\boldsymbol{w}_{\mathcal{H}}$. More concretely, by restricting our attention to flows of average $\boldsymbol{w}_{\mathcal{H}}$-length $\widetilde{O}(n/\phi)$, we only lose a near-constant factor in its value (compared to the optimal maximum flow).

Theorem A.5.6. *Given a flow instance $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ and a ϕ-expander hierarchy $\mathcal{H}$ of $(G, \boldsymbol{c})$ of height η, for any feasible integral $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow $\boldsymbol{f}$ there exists a feasible (not necessarily integral) $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow $\boldsymbol{f}'$ with $|\boldsymbol{f}'| \geq \frac{1}{\eta+1}|\boldsymbol{f}|$ such that $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}') \leq |\boldsymbol{f}'| \cdot O\left(n \cdot \frac{\eta^2 \log n}{\phi}\right)$.*

We will prove Theorem A.5.6 in Section A.5.2. It essentially reduces the maximum flow problem to constructing an expander hierarchy. In Section A.7 we prove such a theorem as follows.

Theorem A.7.1. *There is a randomized algorithm that, given an n-vertex capacitated simple graph $(G, \boldsymbol{c})$, with high probability constructs a $1/n^{o(1)}$-expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of $(G, \boldsymbol{c})$ with $\eta = O(\log n)$ in $n^{2+o(1)}$ time.*

We show how Theorem A.7.1 in combination with Theorem A.5.6 proves our main theorem.

Theorem A.5.7 (Restatement of Theorem A.1.1)**.** *There is a randomized augmenting-path-based algorithm that solves the maximum (s, t)-flow problem on an n-vertex capacitated simple graph with capacities bounded by U in $n^{2+o(1)} \log U$ time with high probability.*

Proof. Via standard capacity scaling techniques (see Section A.9), by paying an $O(\log U)$ multiplicative overhead in the overall running time we can assume all capacities are bounded by n^2 (as in our definition of capacitated graphs in Section A.3).

Therefore, the maximum (s,t)-flow can have value at most n^4. Let $\boldsymbol{\Delta}_s = n^4 \cdot \mathbf{1}_s$ and $\boldsymbol{\nabla}_t = n^4 \cdot \mathbf{1}_t$ be the (s,t)-flow demand. We first invoke Theorem A.7.1 to construct a ϕ-expander hierarchy $\mathcal{H}$ of $(G, \boldsymbol{c})$ for some $\phi = 1/n^{o(1)}$, with height $\eta = O(\log n)$. Then, using the weight function $\boldsymbol{w}_{\mathcal{H}}$ induced by $\mathcal{H}$, we run the push-relabel algorithm (Theorem A.4.1) on the flow instance with $\boldsymbol{w}_{\mathcal{H}}$ and height $h = \Theta\left(\frac{n\eta^2 \log n}{\phi}\right)$, obtaining a flow $\boldsymbol{f}$. We then simply recurse on the residual instance $(G_{\boldsymbol{f}}, \boldsymbol{c}_{\boldsymbol{f}}, \boldsymbol{\Delta}_{s,\boldsymbol{f}}, \boldsymbol{\nabla}_{t,\boldsymbol{f}})$ until there is no augmenting path. By Theorem A.5.6 and Theorem A.4.1(iii), the flow $\boldsymbol{f}$ is an $O(\log n)$-approximation to the maximum (s,t)-flow, which means that the maximum (s,t)-flow value decreases by a factor of $\left(1 - \frac{1}{O(\log n)}\right)$ each time. As such, after $O(\log^2 n)$ iterations the maximum flow value will drop to zero. In each iteration we spend $n^{2+o(1)}$ time constructing $\mathcal{H}$ by Theorem A.7.1 and $\tilde{O}(\frac{n^2}{\phi})$ time (which is also $n^{2+o(1)}$ by our choice of $\phi = n^{-o(1)}$) in the push-relabel algorithm by Theorem A.4.1 and Claim A.5.5. This proves the theorem. □

A.5.2 Existence of Short Flow

To prove Theorem A.5.6, we will instead show the existence of a flow $\boldsymbol{f}'$ with low average path length that routes the *same* demand as $\boldsymbol{f}$ does, at the cost of increasing congestion by $O(\eta)$ factor.

Lemma A.5.8. *Given a flow instance $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ and a ϕ-expander hierarchy $\mathcal{H}$ of G (of height η), for any integral $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow $\boldsymbol{f}$ of congestion $\kappa \in \mathbb{N}$ there exists an equivalent flow $\boldsymbol{f}'$ of congestion $(\eta+1)\kappa$ such that $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}') \leq |\boldsymbol{f}'| \cdot O\left(n \cdot \frac{\eta^2 \log n}{\phi}\right)$.*

Theorem A.5.6 follows from Lemma A.5.8 by simply scaling down $\boldsymbol{f}'$ so that it becomes feasible.

Proof of Theorem A.5.6. By Lemma A.5.8 with $\kappa := 1$, there is an equivalent flow $\boldsymbol{f}''$ with congestion $(\eta + 1)$. The flow $\boldsymbol{f}' := \frac{\boldsymbol{f}''}{\eta+1}$ is a feasible $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow that satisfies $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}') \leq |\boldsymbol{f}'| \cdot O\left(\frac{n\eta^2 \log n}{\phi}\right)$. □

The rest of the section proves Lemma A.5.8. Before diving into the actual proof, we briefly outline the strategy here for better intuition. We start with the not-necessarily short flow $\boldsymbol{f}$, and then "short-cut" some parts of the flow, making the flow shorter at the cost of some congestion. While our arguments here are somewhat algorithmic, we note that for our maximum flow algorithm we only need the *existence* of a short flow, and that the maximum flow algorithm does not need to know the flow $\boldsymbol{f}$ to start with.

To make each flow path short, our goal is to start from the topmost level down, making sure that in each level i and each level-i expander C, the flow path only uses $\widetilde{O}(1/\phi)$ level-i expanding edges in C. If this held for all levels and all expanders within those levels, each flow path would have length $\tilde{O}(n/\phi)$. Therefore, for flow

paths that use a large number of such expanding edges, we have to reroute and short-cut them, using the property of expanders, to reduce the length. Rerouting inevitably incurs congestion in the resulting flow, and if done naïvely the congestion will grow by a multiplicative factor of $1/\phi$ in each level. One key component in our analysis is showing that a more careful way of rerouting actually saves us from this congestion blow-up. Note that we will first prove most of the statements for integral flows, as they admit decomposition into paths that are nice to work with. The statements are then easily extended to fractional flows by simply scaling the flow and capacities up to make them integral.

In the remainder of the section we consider a fixed ϕ-expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of G given to us. By the equivalence between the uncapacitated and capacitated definitions of ϕ-expanding (see Fact A.3.5), *we will assume in our analysis (without loss of generality) that G is a unit-capacitated multi-graph* instead of a capacitated simple graph. Recall from our definition of capacitated graphs in Section A.3 that the capacities are bounded by n^2, and thus after replacing each capacitated edge with multiple parallel edges the graph contains $m \leq n^4$ edges. We need this just so that $\log m = O(\log n)$.

Charging of DAG-edges. We begin by showing that to bound the $\boldsymbol{w}_{\mathcal{H}}$-length of any path in G, it suffices to bound the contribution of expanding edges to its $\boldsymbol{w}_{\mathcal{H}}$-length. To be more flexible for usage also in Section A.6, we consider a slightly more general setting. The following lemma shows that the hierarchy allows us to charge the weight of DAG edges to non-DAG edges.

Lemma A.5.9. *Suppose $G = (V, E)$ is a graph, $D \subseteq E$ is a DAG, $\boldsymbol{\tau}$ a topological order of D, and $\boldsymbol{w}$ is a weight function such that the weight of an edge $e = (u, v)$ satisfies $\boldsymbol{w}(e) \geq |\boldsymbol{\tau}_v - \boldsymbol{\tau}_u|$, with equality if $e \in D$. Then for any path P in G it holds that*

$$\boldsymbol{w}(P \cap D) \leq n + \boldsymbol{w}(P \setminus D).$$

Proof. Suppose we walk along P where $P = (v_1, \ldots, v_k)$. Let $\Phi^{(i)} = \boldsymbol{\tau}_{v_i}$ be the potential that keeps track of how much we proceed in the topological order $\boldsymbol{\tau}$. The *net* potential increase is $\Phi^{(k)} - \Phi^{(1)} \leq n$. Whenever we walk through a DAG-edge e, the potential increases by $\boldsymbol{w}(e)$. So the *total* potential increase is $\sum_{i:\Phi^{(i+1)} > \Phi^{(i)}} \Phi^{(i+1)} - \Phi^{(i)} \geq \boldsymbol{w}(P \cap D)$. The *total* potential decrease is $\sum_{i:\Phi^{(i+1)} < \Phi^{(i)}} \Phi^{(i)} - \Phi^{(i+1)} \leq \boldsymbol{w}(P \setminus D)$ because only non-DAG-edge e may decrease the potential and it decreases by at most $\boldsymbol{w}(e)$. Therefore, we conclude

$$\begin{aligned} n &\geq \Phi^{(k)} - \Phi^{(1)} \\ &= \left(\sum_{i:\Phi^{(i+1)} > \Phi^{(i)}} \Phi^{(i+1)} - \Phi^{(i)} \right) - \left(\sum_{i:\Phi^{(i+1)} < \Phi^{(i)}} \Phi^{(i)} - \Phi^{(i+1)} \right) \\ &\geq \boldsymbol{w}(P \cap D) - \boldsymbol{w}(P \setminus D) \end{aligned}$$

The lemma concludes by rearranging. □

Routing Short Flow in Expanders. Lemma A.5.9 allow us to focus on bounding the length on the expanding edges. In the following sequence of lemmas, we show how to route a flow within a graph so that it uses only few expanding edges. In particular, the lemma below shows that if an edge set F is ϕ-expanding in G, then for any routable demand we can almost reroute it in such a way that each flow path uses at most $O(\log m/\phi)$ edges in F.

Lemma A.5.10. *Consider a routable flow instance $\mathcal{I} = (G, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ for a strongly connected m-edge G in which $F \subseteq E(G)$ is ϕ-expanding. Given any $\varepsilon > 0$, there is a feasible integral $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow $\boldsymbol{f}$ with value $|\boldsymbol{f}| \geq (1-\varepsilon)\|\boldsymbol{\Delta}\|_1$ such that $\sum_{e\in F} \boldsymbol{f}(e) \leq |\boldsymbol{f}| \cdot O\left(\frac{\log m}{\varepsilon\phi}\right)$.*

Proof. Let $\ell := \frac{4\log m}{\varepsilon\phi} = O\left(\frac{\log m}{\varepsilon\phi}\right)$ be the target F-length, and let $\boldsymbol{f}$ be an integral flow in G obtained by repeatedly finding augmenting paths in the residual graph consisting of at most ℓ edges in F and send one unit of flow along them until such paths become non-existent. We get $\sum_{e\in F} \boldsymbol{f}(e) \leq |\boldsymbol{f}| \cdot \frac{4\log m}{\varepsilon\phi}$. If $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1$ then we are done. Otherwise, there is at least one unsaturated source s ($\boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0$) and one unsaturated t ($\boldsymbol{\nabla}(t) > \mathbf{abs}_{\boldsymbol{f}}(t)$). Let $\mathrm{dist}_F(v)$ be the shortest F-distance from an unsaturated source s to vertex v in $G_{\boldsymbol{f}}$, where the F-distance is the minimum F-length over all such paths P. Let $S_i := \{v \in V(G) : \mathrm{dist}_F(v) = i\}$. Note that all unsaturated sinks t must have $\mathrm{dist}_F(t) > \ell$. It suffices to show that there is an $0 \leq i \leq \ell$ such that

$$\left|E_{G_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}})\right| < \varepsilon \cdot \left|E_G(S_{\leq i}, \overline{S_{\leq i}})\right| \tag{A.2}$$

because of the following claim.

Claim A.5.11. *If there is a cut $S_{\leq i}$ satisfying (A.2), then $|\boldsymbol{f}| \geq (1-\varepsilon)\|\boldsymbol{\Delta}\|_1$.*

Proof. The existence of such a cut $S_{\leq i}$ implies $|\boldsymbol{f}| \geq \boldsymbol{f}^{\mathrm{out}}(S_{\leq i}) \geq (1-\varepsilon)|E_G(S_{\leq i}, \overline{S_{\leq i}})|$. Consider the maximum $(\boldsymbol{\Delta}_{\boldsymbol{f}}, \boldsymbol{\nabla}_{\boldsymbol{f}})$-flow $\boldsymbol{f}'$ in $G_{\boldsymbol{f}}$ for which by Fact A.3.2 and that $\mathcal{I}$ is routable implies that $|\boldsymbol{f}| + |\boldsymbol{f}'| = \|\boldsymbol{\Delta}\|_1$. However, by the max-flow min-cut theorem (Fact A.3.4), we have

$$|\boldsymbol{f}'| \leq \boldsymbol{c}_{\boldsymbol{f}}(S_{\leq i}, \overline{S_{\leq i}}) + \boldsymbol{\Delta}_{\boldsymbol{f}}(\overline{S_{\leq i}}) + \boldsymbol{\nabla}_{\boldsymbol{f}}(S_{\leq i}) = \boldsymbol{c}_{\boldsymbol{f}}(S_{\leq i}, \overline{S_{\leq i}})$$

by definition of $S_{\leq i}$. As such, we have $|\boldsymbol{f}'| \leq \frac{\varepsilon}{1-\varepsilon}|\boldsymbol{f}|$ and therefore $|\boldsymbol{f}| \geq (1-\varepsilon)(|\boldsymbol{f}| + |\boldsymbol{f}'|) = (1-\varepsilon)\|\boldsymbol{\Delta}\|_1$. □

Now, assume for contradiction that no $S_{\leq i}$ satisfying (A.2) exists. The following proof is a standard ball-growing argument. Note that by definition of $\mathrm{dist}_F(\cdot)$, it holds that $E_{G_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}}) = E_{G_{\boldsymbol{f}}}(S_i, S_{i+1}) \subseteq F$ for every i. If $\mathrm{vol}_F(S_{\leq \ell/2}) \leq \mathrm{vol}_F(\overline{S_{\leq \ell/2}})$, then we have

$$\begin{aligned}\mathrm{vol}_F(S_{\leq i}) &\geq \mathrm{vol}_F(S_{\leq i-1}) + \left|E_{G_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}})\right| \\ &\overset{(i)}{\geq} \mathrm{vol}_F(S_{\leq i-1}) + \varepsilon\left|E_G(S_{\leq i}, \overline{S_{\leq i}})\right| \overset{(ii)}{\geq} (1+\varepsilon\phi)\cdot \mathrm{vol}_F(S_{\leq i-1})\end{aligned}$$

for $0 < i \leq \ell/2$, where (i) follows from $S_{\leq i}$ not satisfying (A.2) and (ii) follows from F being ϕ-expanding and thus $S_{\leq i}$ is not a ϕ-sparse cut with respect to F. Since $\mathrm{vol}_F(S_0) > 0$, we have

$$\mathrm{vol}_F(S_{\leq \ell/2}) \geq (1+\varepsilon\phi)^{\frac{2\log m}{\varepsilon\phi}} \geq m^2$$

as $(1+x)^{1/x} \geq 2$ for $0 < x \leq 1$, which is a contradiction. Similarly, if instead it is the case that $\mathrm{vol}_F(S_{\leq \ell/2}) > \mathrm{vol}_F(\overline{S_{\leq \ell/2}})$, then

$$\begin{aligned}\mathrm{vol}_F(\overline{S_{\leq i}}) &\geq \mathrm{vol}_F(\overline{S_{\leq i+1}}) + \left|E_{G_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}})\right| \\ &\geq \mathrm{vol}_F(\overline{S_{\leq i+1}}) + \varepsilon\left|E_G(S_{\leq i}, \overline{S_{\leq i}})\right| \geq (1+\varepsilon\phi)\cdot \mathrm{vol}_F(\overline{S_{\leq i+1}})\end{aligned}$$

for $\ell/2 \leq i \leq \ell$. Since $\mathrm{vol}_F(\overline{S_{\leq \ell}}) > 0$, it follows that

$$\mathrm{vol}_F(\overline{S_{\leq \ell/2}}) \geq (1+\varepsilon\phi)^{\frac{2\log m}{\varepsilon\phi}} \geq m^2,$$

which is also a contradiction. □

The lemma above only returns a flow that partially routes a demand. Next, we show that we can fully route any demand by paying a small congestion factor by repeatedly routing the remaining demand. For a subset of edges $F \subseteq E(G)$, we call a demand pair $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is *r-respecting with respect to F* for $r \in \mathbb{N}$ if $\boldsymbol{\Delta}(v), \boldsymbol{\nabla}(v) \leq r \cdot \deg_F(v)$ for each $v \in V$.

Lemma A.5.12. *Given a flow instance $\mathcal{I} = (G, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ where $\|\boldsymbol{\Delta}\|_1 = \|\boldsymbol{\nabla}\|_1$ for a strongly connected m-edge G in which $F \subseteq E(G)$ is ϕ-expanding such that $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is r-respecting with respect to F for $\phi \geq \frac{1}{\mathrm{poly}(m)}$ and $r \leq \mathrm{poly}(m)$, there is a integral $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$-flow $\boldsymbol{f}$ with congestion $O\left(\frac{r}{\phi}\log m\right)$ and value $\|\boldsymbol{f}\| = \|\boldsymbol{\Delta}\|_1$ such that $\sum_{e\in F}\boldsymbol{f}(e) \leq |\boldsymbol{f}| \cdot O\left(\frac{\log m}{\phi}\right)$.*

Proof. By the max-flow min-cut theorem (Fact A.3.4), the r-respecting demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is routable in $G^{(\kappa)}$ for $\kappa := \left\lceil \frac{r}{\phi} \right\rceil$.[20] Observe that $F^{(\kappa)}$ is ϕ-expanding in $G^{(\kappa)}$. As such, by Lemma A.5.10 with $\varepsilon := 1/2$ there is an integral flow $\boldsymbol{f}_1$ in $G^{(\kappa)}$ such that $\|\boldsymbol{f}_1\| \geq \frac{1}{2}\|\boldsymbol{\Delta}\|_1$ and $\sum_{e\in F^{(\kappa)}}\boldsymbol{f}_1(e) \leq |\boldsymbol{f}_1|\cdot O\left(\frac{\log m}{\phi}\right)$ since $\kappa \leq \mathrm{poly}(m)$. The residual demand $(\boldsymbol{\Delta}_{\boldsymbol{f}_1}, \boldsymbol{\nabla}_{\boldsymbol{f}_1})$ satisfies $\|\boldsymbol{\Delta}_{\boldsymbol{f}_1}\|_1 \leq \frac{1}{2}\|\boldsymbol{\Delta}\|_1$ and is clearly also r-respecting with respect to F and thus routable in $G^{(\kappa)}$. Applying Lemma A.5.10 again with $\varepsilon := \frac{1}{2}$ on

[20] For any cut $S \subseteq V$ in $G^{(\kappa)}$, we have $|E_{G^{(\kappa)}}(S, \overline{S})| \geq r \cdot \min\{\mathrm{vol}_F(S), \mathrm{vol}_F(\overline{S})\} \geq \min\{\boldsymbol{\Delta}(S), \boldsymbol{\nabla}(\overline{S})\}$ by definition. By the max-flow min-cut theorem (Fact A.3.4), the maximum flow in $G^{(\kappa)}$ has value

$$\min_S \left|E_{G^{(\kappa)}}(S, \overline{S})\right| \geq \min_S \left\{\min\{\boldsymbol{\Delta}(S), \boldsymbol{\nabla}(\overline{S})\} + \boldsymbol{\Delta}(\overline{S}) + \boldsymbol{\nabla}(S)\right\} \geq \|\boldsymbol{\Delta}\|_1 = \|\boldsymbol{\nabla}\|_1.$$

Therefore, $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is routable in $G^{(\kappa)}$.

the $(G^{(\kappa)}, \boldsymbol{\Delta}_{\boldsymbol{f}_1}, \boldsymbol{\nabla}_{\boldsymbol{f}_1})$, we get an integral flow $\boldsymbol{f}_2$ in $G^{(\kappa)}$ such that $\|\boldsymbol{f}_2\|_1 \geq \frac{1}{2}\|\boldsymbol{\Delta}_{\boldsymbol{f}_1}\|$ and $\sum_{e\in F^{(\kappa)}} \boldsymbol{f}_2(e) \leq |\boldsymbol{f}_2| \cdot O\left(\frac{\log m}{\phi}\right)$. Because $\|\boldsymbol{\Delta}\|_1 \leq \text{poly}(m)$, repeating this argument $O(\log m)$ times, we get $O(\log m)$ integral flows $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{O(\log m)}$ such that the sum of them $\boldsymbol{f} \coloneqq \boldsymbol{f}_1 + \cdots + \boldsymbol{f}_{O(\log m)}$ has value $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1$ and $\sum_{e\in F^{(\kappa)}} \boldsymbol{f}(e) \leq |\boldsymbol{f}| \cdot O\left(\frac{\log m}{\phi}\right)$ with congestion $O(\log m)$ in $G^{(\kappa)}$, which can be mapped back to an integral flow in G with congestion $O(\kappa \log m) = O\left(\frac{r}{\phi}\log m\right)$ with the same guarantee. □

Rerouting Long Flow to Short Flow. If we directly use Lemma A.5.12 for rerouting each flow path, we might get a congestion blow-up of $\widetilde{\Theta}(1/\phi)$ which is too expensive. The crucial idea to avoid this is as follows: for each long flow path, we reroute the flow starting at the set of first $\widetilde{O}(1/\phi)$ edges of the path to the set of last $\widetilde{O}(1/\phi)$ edges. This idea leads to cancellation in congestion and allows us to control the congestion blow-up to be at most $(1 + 1/\eta)$ factor, which is only $O(\eta)$ factor after accumulation over all η levels.

We begin by defining what we mean by *rerouting*. Given an integral flow $\boldsymbol{f}$ decomposable into paths $P_1, \ldots, P_k$, we can *reroute* $\boldsymbol{f}$ at (s_i, t_i) for each $1 \leq i \leq k$, where s_i and t_i are vertices on P_i (with s_i occurring before t_i) with a flow $\boldsymbol{f}_{\text{route}}$ routing the demand

$$\boldsymbol{\Delta}(v) \coloneqq |\{1 \leq i \leq k : s_i = v\}|, \qquad \boldsymbol{\nabla}(v) = |\{1 \leq i \leq k : t_i = v\}|$$

getting the flow $\widetilde{\boldsymbol{f}}$ given by $\widetilde{\boldsymbol{f}}(e) \coloneqq \boldsymbol{f}(e) + \boldsymbol{f}_{\text{route}}(e) - \sum_{i=1}^{k} \boldsymbol{f}_{P_i[s_i,t_i]}(e)$, where $\boldsymbol{f}_{P_i[s_i,t_i]}$ is the notation for a flow that sends one unit flow along the path $P_i[s_i, t_i]$. We note that we *do* ***not*** *use multi-commodity flow* when rerouting; that is, $\boldsymbol{f}_{\text{route}}$ does not necessarily consist of (s_i, t_i)-paths. Indeed, for our purposes we only need that each s_i is paired up with some t_j, i.e., that $\boldsymbol{f}_{\text{route}}$ routes the same demand as the flow $\sum_{i=1}^{k} \boldsymbol{f}_{P_i[s_i,t_i]}$.

Observation A.5.13. *The following facts about such a rerouted flow hold.*

(1) $\widetilde{\boldsymbol{f}}$ *routes the same demands as* $\boldsymbol{f}$ *does, i.e.,* $\widetilde{\boldsymbol{f}}$ *and* $\boldsymbol{f}$ *are equivalent.*

(2) If $\boldsymbol{f}_{\text{route}}$ *is a flow in a subgraph* H *of* G*, then* $\widetilde{\boldsymbol{f}}(e) \leq \boldsymbol{f}(e)$ *for all* $e \notin E(H)$*.*

(3) If $\boldsymbol{f}_1 + \boldsymbol{f}_2$ *has congestion* κ *and* $\boldsymbol{f}_1$ *is rerouted by a flow* $\boldsymbol{f}_{\text{route}}$ *with congestion* κ'*, resulting in* $\widetilde{\boldsymbol{f}_1}$*, then the flow* $\widetilde{\boldsymbol{f}_1} + \boldsymbol{f}_2$ *has congestion* $\kappa + \kappa'$*.*

Let $c_{A.5.12} \in \mathbb{N}$ be the constant hidden in the $O(\cdot)$ notation of the congestion guarantee of Lemma A.5.12. In other words, the flow from Lemma A.5.12 has congestion at most $\frac{r}{\phi} \cdot c_{A.5.12} \log m$. To recall, η is the height of the given hierarchy $\mathcal{H}$. Now we are ready to prove our main rerouting lemma that incurs only a very small congestion blow-up.

Lemma A.5.14. *Given a flow $\boldsymbol{f}$ with congestion κ in G, a target level i, and a level-i expander C of G, there is an equivalent flow $\boldsymbol{f}'$ in G with congestion $\lceil\kappa\rceil\left(1+\frac{1}{\eta}\right)$ such that*

$$\sum_{e\in F}\boldsymbol{f}'(e)\leq|\boldsymbol{f}'|\cdot O\left(\frac{\eta\log n}{\phi}\right),$$

where $F\coloneqq X_i\cap E(C)$ is the level-i expanding edge set in C. Additionally, it holds that $\boldsymbol{f}'(e)\leq\boldsymbol{f}(e)$ for all $e\notin E(C)$.

Proof. Let us first suppose that $\boldsymbol{f}$ is an integral flow (therefore we may assume $\kappa\in\mathbb{N}$), and let $\xi\coloneqq\left\lceil\frac{\eta\cdot c_{A.5.12}\log m}{\phi}\right\rceil\in\mathbb{N}$. Let $\mathcal{P}_{\boldsymbol{f}}$ be a decomposition of $\boldsymbol{f}$ into flow paths. Let $\mathcal{P}_{\text{long}}\coloneqq\{P\in\mathcal{P}_{\boldsymbol{f}}:|P\cap F|\geq 2\xi\}$ be the paths which are long with respect to F, i.e., uses at least 2ξ edges in F (note that we may assume that the paths P are simple and thus cannot use the same edge in F multiple times). For each $P\in\mathcal{P}_{\text{long}}$, let $S_P\coloneqq\left(s_P^{(1)},\cdots,s_P^{(\xi)}\right)$ be the endpoints of the first ξ edges from F on P. Similarly, let $T_P\coloneqq\left(t_P^{(1)},\ldots,t_P^{(\xi)}\right)$ be the endpoints of the last ξ edges from F on P. Let $\boldsymbol{f}_{\text{long}}\coloneqq\sum_{P\in\mathcal{P}_{\text{long}}}\boldsymbol{f}_P$ and $\boldsymbol{f}_{\text{short}}\coloneqq\boldsymbol{f}-\boldsymbol{f}_{\text{long}}=\sum_{P\in\mathcal{P}_{\boldsymbol{f}}\setminus\mathcal{P}_{\text{long}}}\boldsymbol{f}_P$ be the flow corresponding to long and short flow paths, respectively. Let $\boldsymbol{f}_{\text{long}}^{(\xi)}\coloneqq\boldsymbol{f}_{\text{long}}\cdot\xi$ be a flow in G and $\mathcal{P}_{\text{long}}^{(\xi)}$ be the decomposition of $\boldsymbol{f}_{\text{long}}^{(\xi)}$ corresponding to $\mathcal{P}_{\text{long}}$, i.e., $\mathcal{P}_{\text{long}}^{(\xi)}\coloneqq\bigcup_{P\in\mathcal{P}_{\text{long}}}\left\{P_1^{(\xi)},\ldots,P_\xi^{(\xi)}\right\}$ where $P_i^{(\xi)}$ is the i-th duplicate of the path $P\in\mathcal{P}_{\text{long}}$.

We now reroute $\boldsymbol{f}_{\text{long}}^{(\xi)}$ at $\left\{\left(s_P^{(i)},t_P^{(i)}\right)\right\}_{P_i^{(\xi)}\in\mathcal{P}_{\text{long}}^{(\xi)}}$. In other words, for the i-th copy of $P\in\mathcal{P}_{\text{long}}$, we attempt to reroute it from the i-th edge in S_P to the i-th edge in T_P. This is the main idea which allow us to avoid the congestion blow-up, since, although $\boldsymbol{f}_{\text{long}}^{(\xi)}$ has congestion $\xi\kappa$, the demand $(\boldsymbol{\Delta},\boldsymbol{\nabla})$ corresponding to this rerouting is κ-respecting on F. This is since each start-vertex $s_P^{(i)}$ and end-vertex $t_P^{(i)}$ in the rerouting can be charged (a single time) to the corresponding edge in the flow path P, and $\boldsymbol{f}$ has congestion κ. Therefore by Lemma A.5.12, $\boldsymbol{f}_{\text{long}}^{(\xi)}$ can be routed in C with congestion $\frac{\kappa}{\phi}\cdot c_{A.5.12}\log m$ by a flow $\boldsymbol{f}_{\text{route}}$. Let $\widetilde{\boldsymbol{f}_{\text{long}}^{(\xi)}}$ be the rerouted $\boldsymbol{f}_{\text{long}}^{(\xi)}$. It then follows that the flow

$$\boldsymbol{f}'\coloneqq\boldsymbol{f}_{\text{short}}+\frac{\widetilde{\boldsymbol{f}_{\text{long}}^{(\xi)}}}{\xi}$$

routes the same demand as $\boldsymbol{f}$ does by Observation A.5.13(1) and has congestion

$$\kappa+\frac{\frac{\kappa}{\phi}\cdot c_{A.5.12}\log m}{\xi}=\kappa\left(1+\frac{1}{\eta}\right)$$

by Observation A.5.13(3). The total amount of flow on F-edges can be bounded by

$$\begin{aligned}\sum_{e\in F}\boldsymbol{f}'(e) &= \sum_{e\in F}\boldsymbol{f}_{\text{short}}(e) + \frac{\sum_{e\in F}\widetilde{\boldsymbol{f}^{(\xi)}_{\text{long}}}(e)}{\xi}\\ &\le |\boldsymbol{f}_{\text{short}}|\cdot 2\xi + |\boldsymbol{f}_{\text{long}}|\cdot 2\xi + |\boldsymbol{f}_{\text{long}}|\cdot O\left(\frac{\log m}{\phi}\right)\\ &\le |\boldsymbol{f}'|\cdot O\left(\frac{\eta\log n}{\phi}\right),\end{aligned}$$

where we use the fact that the rerouting happens at the first ξ and the last ξ edges on flow paths in $\mathcal{P}_{\text{long}}$. The property that $\boldsymbol{f}'(e)\le\boldsymbol{f}(e)$ for all $e\notin E(C)$ also follows from Observation A.5.13(2).

If instead the flow $\boldsymbol{f}$ is $\frac{1}{z}$-integral for some $z\in\mathbb{N}$, then we may assume $\kappa\in\frac{1}{z}\cdot\mathbb{N}$ and the demand $(\boldsymbol{\Delta},\boldsymbol{\nabla})$ it routes to be in $\left(\frac{1}{z}\cdot\mathbb{N}\right)^V$. We can then treat $\boldsymbol{f}$ as an integral flow in $G^{(z)}$ routing demand $(z\cdot\boldsymbol{\Delta}, z\cdot\boldsymbol{\nabla})$ with congestion $\lceil\kappa\rceil$, i.e., for each edge e, we put a total of $\boldsymbol{f}(e)\cdot z\le\kappa z$ units of flow on the duplicates of e in $G^{(z)}$, distributed evenly among the z duplicates so that each of them receives at most $\lceil\kappa\rceil$ units of flow. Applying the same argument as before in $G^{(z)}$ proves the lemma for this case.

Finally, note that the rerouted flow $\boldsymbol{f}'$ is $\frac{1}{z\xi}$-integral so $\boldsymbol{f}'\in\mathbb{Q}^E_{\ge 0}$ (recall that our definition of flow requires rational values). □

Corollary A.5.15. *Given a flow $\boldsymbol{f}$ with congestion κ in G and a target level i, there is a flow $\boldsymbol{f}'$ in G routing the same demand as $\boldsymbol{f}$ does with congestion $\lceil\kappa\rceil\left(1+\frac{1}{\eta}\right)$ such that*

$$\sum_{e\in X_i}\boldsymbol{f}'(e)\boldsymbol{w}_{\mathcal{H}}(e)\le|\boldsymbol{f}'|\cdot O\left(n\cdot\frac{\eta\log n}{\phi}\right).$$

Additionally, it holds that $\boldsymbol{f}'(e)\le\boldsymbol{f}(e)$ for all $e\in X_{>i}$.

Proof. The corollary simply follows by applying Lemma A.5.14 to every level-i expander in an arbitrary order, using the fact that the vertex-sizes of the level-i expanders sum up to n and a level-i expanding edge e has weight $\boldsymbol{w}_{\mathcal{H}}(e)\le|C|$ for C being the level-i expander e belongs to. □

Lemma A.5.8 can now be proved.

Lemma A.5.8. *Given a flow instance $\mathcal{I}=(G,\boldsymbol{c},\boldsymbol{\Delta},\boldsymbol{\nabla})$ and a ϕ-expander hierarchy $\mathcal{H}$ of G (of height η), for any integral $(\boldsymbol{\Delta},\boldsymbol{\nabla})$-flow $\boldsymbol{f}$ of congestion $\kappa\in\mathbb{N}$ there exists an equivalent flow $\boldsymbol{f}'$ of congestion $(\eta+1)\kappa$ such that $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}')\le|\boldsymbol{f}'|\cdot O\left(n\cdot\frac{\eta^2\log n}{\phi}\right)$.*

Proof. Let $\boldsymbol{f}_{\eta+1}\coloneqq\boldsymbol{f}$ be the given integral flow with congestion $\kappa_{\eta+1}=\kappa$. From $i=\eta$ to 1, we apply Corollary A.5.15 on $\boldsymbol{f}_{i+1}$ and κ_{i+1} on target level i, getting a

flow $\boldsymbol{f}_i$ with congestion $\kappa_i \leq \lceil \kappa_{i+1} \rceil \left(1 + \frac{1}{\eta}\right)$. By induction, it is easy to see that $\kappa_i \leq \kappa \cdot (\eta + 2 - i)$. The returned flow $\boldsymbol{f}''$ is then set to $\boldsymbol{f}_1$, which has congestion $\kappa(\eta+1)$, with the property that

$$\sum_{e \in E \setminus D} \boldsymbol{f}''(e)\boldsymbol{w}_{\mathcal{H}}(e) \leq \eta \cdot |\boldsymbol{f}''| \cdot O\left(n \cdot \frac{\eta \log n}{\phi}\right)$$

using that each rerouting does not affect (or can only decrease) flows on higher-level edges by Observation A.5.13(2). The lemma then follows from Lemma A.5.9 which asserts that the weights of DAG edges on a path are bounded by the weights of expanding edges on it, up to an additive factor of n which is dominated. □

The above lemma shows that there is a *fractional* flow which is also short. While this is good enough to guarantee that our push relabel algorithm returns an approximate flow, we note in the following corollary that, by paying another $\log(n)$-factor in congestion, we can assume the short flow is *integral*. This observation will be useful later in Section A.6.2.

Corollary A.5.16. *Given a flow instance $\mathcal{I} = (G, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ routable with congestion $\kappa \in \mathbb{N}$ in a graph G equipped with a ϕ-expander hierarchy $\mathcal{H}$ of height η, there is an integral flow $\boldsymbol{f}$ routing $\mathcal{I}$ with congestion $O(\kappa\eta \log n)$ such that $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}) = |\boldsymbol{f}| \cdot O\left(n \cdot \frac{\eta^2 \log n}{\phi}\right)$.*

Proof. The existence of a fractional such flow is given by Lemma A.5.8. By Corollary A.4.4, we get a short integral flow $\boldsymbol{f}_1$ routing $\frac{1}{6}$ fraction of $\|\boldsymbol{\Delta}\|_1$. On the residual demand $(\boldsymbol{\Delta}_{\boldsymbol{f}}, \boldsymbol{\nabla}_{\boldsymbol{f}})$ we may apply the same argument again in G (not in $G_{\boldsymbol{f}_1}$) and get a short integral $\boldsymbol{f}_2$ routing $\frac{1}{6}$ fraction of $\|\boldsymbol{\Delta}_{\boldsymbol{f}_1}\|$ Repeating this $O(\log n)$ times until the demand becomes empty, we get $O(\log n)$ flows $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{O(\log n)}$ in G, each with congestion $O(\kappa\eta)$ and $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}_i) = O\left(|\boldsymbol{f}_i| \cdot \frac{n\eta^2 \log n}{\phi}\right)$, for which $\boldsymbol{f} \coloneqq \boldsymbol{f}_1 + \cdots + \boldsymbol{f}_{O(\log n)}$ routes $\mathcal{I}$. We also have $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}) \leq \boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}_1) + \cdots + \boldsymbol{w}_{\mathcal{H}}\left(\boldsymbol{f}_{O(\log n)}\right) \leq O\left(|\boldsymbol{f}| \cdot \frac{n\eta^2 \log n}{\phi}\right)$, proving the corollary. □

A.6 The Sparse-Cut Algorithm

A central building block in constructing expander decompositions or even expander hierarchies in general is to either solve a flow problem or find a sparse cut in the graph. In this section we provide such a subroutine using our push-relabel algorithm. Our algorithm to build the expander hierarchy (in Section A.7) will heavily rely on the following theorem which we prove here.

Theorem A.6.1. *Given a diffusion instance $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ on a strongly connected n-vertex graph G, a ϕ-expander hierarchy $\mathcal{H}$ of $(G \setminus F, \boldsymbol{c})$ of height η, and some $\kappa \in \mathbb{N}$ with $1/\phi, \kappa \leq n$, there is an $\widetilde{O}(n^2 \cdot \frac{\kappa\eta^4}{\phi^2})$ time algorithm* SparseCut$(\mathcal{I}, \kappa, F, \mathcal{H})$

that finds a flow $\boldsymbol{f}$ *with congestion* κ *and, if* $|\boldsymbol{f}| < \|\boldsymbol{\Delta}\|_1$*, a cut* $\emptyset \neq S \subsetneq V$ *with* $\mathbf{abs}_{\boldsymbol{f}}(S) = \boldsymbol{\nabla}(S)$ *and* $\mathbf{ex}_{\boldsymbol{f}}(S) = \mathbf{ex}_{\boldsymbol{f}}(V)$ *such that*

$$\boldsymbol{c}(E_G(S, \overline{S})) \leq \frac{O(|\boldsymbol{f}|) + \min\{\mathrm{vol}_{F,\boldsymbol{c}}(S), \mathrm{vol}_{F,\boldsymbol{c}}(\overline{S})\}}{\kappa}. \tag{A.3}$$

Remark A.6.2. When $F = \emptyset$ and $\kappa = 1$, Theorem A.6.1 is an $O(1)$-approximate maximum flow algorithm because it either routes all the source, otherwise there is a cut S where $\boldsymbol{c}(E_G(S, \overline{S})) = O(|\boldsymbol{f}|)$, which certifies that $\boldsymbol{f}$ is an $O(1)$-approximation. One can view this theorem as a generalization of our approximate maximum flow algorithm, as explained in the proof of Section A.5 and Theorem A.5.7, where we do not quite have a ϕ-expander hierarchy of the full graph, but only of $G \setminus F$ for some edge set F. The quality of the flow (and cut) we can find here will depend on the edge set F (see the vol_F-terms in the theorem statement). As we will see later in Section A.7, the guarantees here are good enough for the sparse-cut subroutines we need when building the expander hierarchy: in particular, using Theorem A.6.1 we can either certify that F is $\widetilde{\Theta}(\frac{1}{\kappa})$-expanding in G or else find a sparse cut with respect to F.

The Algorithm. We first describe the algorithm for Theorem A.6.1 whose pseudocode is given in Algorithm A.2.

Algorithm A.2: SparseCut$(\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla}), \kappa, F, \mathcal{H})$

Let $h \coloneqq \left\lceil \frac{4\eta^4 \cdot c_{A.6.5} \cdot \log^7 n \cdot \kappa}{\phi^2} \cdot n \right\rceil$, $\boldsymbol{c}^\kappa \coloneqq \kappa \cdot \boldsymbol{c}$, and

$$\boldsymbol{w}_G(e) \coloneqq \begin{cases} \boldsymbol{w}_{\mathcal{H}}(e) & \text{for } e \in E \setminus F \text{ (see (A.1))} \\ n & \text{for } e \in F \end{cases}.$$

Run PushRelabel$(G, \boldsymbol{c}^\kappa, \boldsymbol{\Delta}, \boldsymbol{\nabla}, \boldsymbol{w}_G, h)$ (Theorem A.4.1) to get a flow $\boldsymbol{f}$.

if $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1$ **then return** $\boldsymbol{f}$

else

  Let $\boldsymbol{w}_{\boldsymbol{f}}$ be $\boldsymbol{w}_G$ extended to $G_{\boldsymbol{f}}$, except set $\boldsymbol{w}_{\boldsymbol{f}}(\overrightarrow{e}) \coloneqq 0$ for $e \in D_{\mathcal{H}}$.

  Let $S_0 = \{s \in V : \boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0\}$.

  Compute $\boldsymbol{w}_{\boldsymbol{f}}$-distance levels $S_i \coloneqq \left\{v \in V : \mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(S_0, v) = i\right\}$ in the residual graph $G_{\boldsymbol{f}}$.

  return $\boldsymbol{f}$ and the cut $(S_{\leq i}, \overline{S_{\leq i}})$ minimizing $\boldsymbol{c}^\kappa_{\boldsymbol{f}}(E_{G_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}})) - \min\{\mathrm{vol}_F(S_{\leq i}), \mathrm{vol}_F(\overline{S_{\leq i}})\}$.

The main idea is to run our push-relabel algorithm to try to route the demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$. If it fails to find a large enough flow, we will show how to extract a "sparse cut" from the residual graph. In order to run our push-relabel algorithm (Theorem A.4.1 and Algorithm A.1), we need to supply it with a weight function $\boldsymbol{w}$. However, we do not yet have a ϕ-expander hierarchy of the whole graph G, but only

of $G \setminus F$. A natural idea is to extend the weight function $\boldsymbol{w}_{\mathcal{H}}$ to all of G, assigning edges in F a large weight.

Let $\mathcal{H}$ be the given hierarchy for $G \setminus F$. Let

$$h \coloneqq \left\lceil \frac{4\eta^4 \cdot c_{A.6.5} \cdot \log^7 n \cdot \kappa}{\phi^2} \cdot n \right\rceil = O\left(\frac{n \cdot \eta^4 \log^7 n \cdot \kappa}{\phi^2}\right), \tag{A.4}$$

for a constant $c_{A.6.5}$ that will be defined later in (A.6). Let $\boldsymbol{c}^\kappa \coloneqq \kappa \cdot \boldsymbol{c}$ as in Algorithm A.2. We apply Theorem A.4.1 on the flow instance $\mathcal{I}^\kappa \coloneqq (G, \boldsymbol{c}^\kappa, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ to height h and weight function $\boldsymbol{w}_G$ where $\boldsymbol{w}_G(e) \coloneqq \boldsymbol{w}_{\mathcal{H}}(e)$ for $e \in E \setminus F$ and $\boldsymbol{w}_G(e) \coloneqq n$ for $e \in F$. Let $\boldsymbol{f}$ be the flow returned by Theorem A.4.1. If $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1$, we are done. Otherwise $|\boldsymbol{f}| < \|\boldsymbol{\Delta}\|_1$, in which case by Theorem A.4.1(i) we have $\mathrm{dist}^{\boldsymbol{w}_G}_{G_{\boldsymbol{f}}}(s,t) > 3h$ for any $\boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0$ and $\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$. In this case we need to find a sparse cut.

Running Time. The weight function $\boldsymbol{w}_G$ can be computed in $O(m\eta)$ time by Lemma A.5.3. The distance layers can be computed with a standard shortest path algorithm (e.g., Dijkstra's algorithm [Dij59]) in $\tilde{O}(m)$ time. By Theorem A.4.1 and Claim A.5.5, the running time of the PushRelabel() call is $\widetilde{O}\left(n^2 \cdot \frac{\kappa\eta^4}{\phi^2}\right)$.

Analysis in a Unit-Capacitated Multi-Graph. By the equivalence between the uncapacitated and capacitated definitions of ϕ-expanding (see Fact A.3.5), we will assume (without loss of generality), for the remainder of this section, in our analysis that G is a unit-capacitated multi-graph instead of a capacitated simple graph. That is, $\boldsymbol{c} = \mathbf{1}$ and $\boldsymbol{c}^\kappa = \kappa \cdot \mathbf{1}$. Recall that the capacities are bounded by n^2 (Section A.3), and thus after replacing each capacitated edge with multiple parallel edges the graph contains $m \le n^4$ edges.

Finding a Sparse Cut. To locate a sparse cut when $G \setminus F$ is the empty graph (that is, when we want to build the first level of expander decomposition) the following strategy is standard (see e.g. [HRW17; SW19]) and sufficient for us: *Let* $S_0 = \{s : \boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0\}$ *and compute the distance layers* $S_i = \{v : \mathrm{dist}^{\boldsymbol{w}_G}_{G_{\boldsymbol{f}}}(S_0, v) = i\}$. *Now at least one of the level cuts* $E_G(S_{\le i}, \overline{S_{\le i}})$ *must be sparse.* The proof of this strategy follows from a simple ball-growing argument.

Unfortunately, even when the underlying graph $G \setminus F$ is a DAG, the above strategy fails. The problem is that there might be too many DAG-edges crossing the level cuts. To solve this, we will modify the weight function slightly by setting all forward DAG edges to have weight 0. In particular, we let $\boldsymbol{w}_{\boldsymbol{f}}$ be the weight function on $E(G_{\boldsymbol{f}})$, where $\boldsymbol{w}_{\boldsymbol{f}}(e) = \boldsymbol{w}_G(e)$ for all e except for $\boldsymbol{w}_{\boldsymbol{f}}(\overrightarrow{e}) = 0$ for $e \in D_{\mathcal{H}}$.

As we will see in the remainder of this section, if we compute the distance layers with respect to $\boldsymbol{w}_{\boldsymbol{f}}$, at least one of the level cuts must be sparse. This means that the algorithm to find such a sparse cut is quite simple: just compute the distances, and output the sparsest of the level cuts. While the algorithm itself is simple,

showing the *existence* of such a sparse level cut turns out to be nontrivial. There are essentially three types of edges we want to argue are sparse in most level cuts.

DAG edges of $\mathcal{H}$. The modification to the weight function makes it so that only DAG edges used in the flow can be in a level-cut, of which there are on average $O(|\boldsymbol{f}|)$ crossing each level cut.

Edges in F. These edges can be handled by a ball-growing argument (see proof of Lemma A.6.4), similar to the case when constructing a single level expander decomposition. This shows that most level cuts $(S_{\leq i}, \overline{S_{\leq i}})$ have at most $\min\{\mathrm{vol}_F(S_{\leq i}), \mathrm{vol}_F(\overline{S_{\leq i}})\}$ edges from F crossing them. However, our modification of setting some weights to zero might have reduced the number of layers. So we must argue that we still have enough level cuts left in the graph, or, equivalently, we want the distance from any source to any sink in the residual graph to still be $\Omega(h)$. We argue this in Lemma A.6.3.

Expanding edges of $\mathcal{H}$. These are arguably the trickiest edges to handle and is thus the focus of the majority of our analysis. We want to argue that most level cuts have few expanding edges of $\mathcal{H}$ in them. In the original graph G, each expander in $\mathcal{H}$ has a low diameter. If we can say that this is also the case in the residual graph $G_{\boldsymbol{f}}$, we can argue that each expander will only span a few level cuts, so most level cuts do not have any expanding edges at all. Using the properties of expanders and how the residual graph is constructed by reversing short augmenting paths, we show something in this direction. We prove in Section A.6.2 a "low-diameter expander pruning lemma" which states that a large portion of each expander in $\mathcal{H}$ remain intact and of low diameter also in the residual graph $G_{\boldsymbol{f}}$, and that there are only a few "pruned" edges which cannot contribute too much to the size of all level cuts.

The Modified Weight Function. We begin by showing that although the weights of some edges are set to 0 in $\boldsymbol{w_f}$, the distance in the residual graph remains large. Overloading notation, let

$$\mathrm{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}}}(v) \coloneqq \min_{\boldsymbol{\Delta_f}(s)>0} \mathrm{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}}}(s, v).$$

Lemma A.6.3. *If* $\mathrm{dist}^{\boldsymbol{w}_G}_{G_{\boldsymbol{f}}}(v) > 3h$, *then* $\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(v) > h$.

Proof. Consider a vertex v and let P be the shortest path with respect to $\boldsymbol{w_f}$ from an unsaturated source to v in $G_{\boldsymbol{f}}$. Thus, the $\boldsymbol{w_f}$-weight of P is $\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(v) = \boldsymbol{w}_G(P \setminus \overrightarrow{D})$ because the weight $\boldsymbol{w_f}$ is the same as $\boldsymbol{w}_G$ except that the weights of all forward DAG-edges are set to zero.

We note that $\boldsymbol{w}_G$ satisfies the assumption of Lemma A.5.9 (in the graph $G_{\boldsymbol{f}}$, with the DAG $\overrightarrow{D}$), i.e., that $\boldsymbol{w}_G(e) \geq |\boldsymbol{\tau}_u - \boldsymbol{\tau}_v|$ for any edge $e = (u, v)$ since $\boldsymbol{w}_G(e) = n$

for $e \in F$ and otherwise it follows from the definition (A.1) of $\boldsymbol{w}_{\mathcal{H}}$. Hence we have

$$\boldsymbol{w}_G(P \cap \overrightarrow{D}) \leq n + \boldsymbol{w}_G(P \setminus \overrightarrow{D}) = n + \mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(v).$$

Since $\mathrm{dist}^{\boldsymbol{w}_G}_{G_{\boldsymbol{f}}}(v)$ is the shortest distance to v (with respect to $\boldsymbol{w}_G$), we have

$$3h < \mathrm{dist}^{\boldsymbol{w}_G}_{G_{\boldsymbol{f}}}(v) \leq \boldsymbol{w}_G(P) = \boldsymbol{w}_G(P \cap \overrightarrow{D}) + \boldsymbol{w}_G(P \setminus \overrightarrow{D}) \leq n + 2\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(v).$$

Rearranging, we see that $\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(v) > \frac{3h-n}{2} \geq h$, as $h \geq n$. □

Level Cuts. Let

$$S_i \coloneqq \left\{ v \in V : \mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{G_{\boldsymbol{f}}}(v) = i \right\}$$

be the distance levels in the residual graph with respect to this reduced weight function $\boldsymbol{w}_{\boldsymbol{f}}$. By Lemma A.6.3, we know that $S_{\leq h} \neq V$. Theorem A.6.1 now directly follows from the below key lemma that establishes the existence of a sparse level cut. In the remainder of the section we prove Lemma A.6.4.

Lemma A.6.4. *There exists a level cut $S_{\leq i}$ with $0 \leq i \leq h$ such that*

$$\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(E_{G_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}})) \leq O(|\boldsymbol{f}|) + \min\{\mathrm{vol}_F(S_{\leq i}), \mathrm{vol}_F(\overline{S_{\leq i}})\}. \tag{A.5}$$

Proof of Theorem A.6.1. We take $(S_{\leq i}, \overline{S_{\leq i}})$ as the output cut $(S, \overline{S})$. By definition, Theorem A.4.1(i), and Lemma A.6.3, we have $S_0 = \{s : \mathbf{ex}_{\boldsymbol{f}}(s) > 0\}$ and $S_{\leq h} \cap \{t : \mathbf{abs}_{\boldsymbol{f}}(t) < \boldsymbol{\nabla}(t)\} = \emptyset$, and therefore $\mathbf{ex}_{\boldsymbol{f}}(S_{\leq i}) = \mathbf{ex}_{\boldsymbol{f}}(V)$ and $\mathbf{abs}_{\boldsymbol{f}}(S_{\leq i}) = \boldsymbol{\nabla}(S_{\leq i})$ hold.

What remains is to show that a cut $(S, \overline{S})$ satisfying (A.5) (which, by Lemma A.6.4 our algorithm will find whenever $|\boldsymbol{f}| < \|\boldsymbol{\Delta}\|_1$) also satisfies the output requirement (A.3) of Theorem A.6.1, i.e., $\boldsymbol{c}^{\kappa}(E_G(S, \overline{S})) \leq O(|\boldsymbol{f}|) + \min(\mathrm{vol}_F(S), \mathrm{vol}_F(\overline{S}))$. Indeed this is the case since $\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(E_{G_{\boldsymbol{f}}}(S, \overline{S})) = \boldsymbol{c}^{\kappa}(E_G(S, \overline{S})) - \boldsymbol{f}^{\mathrm{out}}(S) \geq \boldsymbol{c}^{\kappa}(E_G(S, \overline{S})) - |\boldsymbol{f}|$ by Fact A.3.1. □

A.6.1 Existence of Sparse Level Cuts

To prove Lemma A.6.4, we show that each expander, while in the residual graph, has a relatively large portion that still has a low diameter. Fixing a level ℓ in the hierarchy, let $\left\{C^{(1)}_{\ell}, C^{(2)}_{\ell}, \ldots, C^{(k)}_{\ell}\right\}$ be the strongly connected components of $(G \setminus F) \setminus X_{>\ell}$. That is, $C^{(i)}_{\ell}$ is a level-ℓ expander and denote by $X^{(i)}_{\ell} = X_{\ell} \cap E(C^{(i)}_{\ell})$ the set of level-ℓ expanding edges in $C^{(i)}_{\ell}$.

We now argue that except for a small subset of "pruned" edges $P^{(i)}_{\ell}$, the edges of the expander remain well-connected and more importantly stay relatively close to each other.

Lemma A.6.5. *There exists a subset $P^{(i)}_{\ell} \subseteq X^{(i)}_{\ell}$ such that*

(1) *for each pair* $e_1, e_2 \in X_\ell^{(i)} \setminus P_\ell^{(i)}$, *we have* $\operatorname{dist}_{G_{\boldsymbol{f}}}^{\boldsymbol{w}_{\boldsymbol{f}}}(\overrightarrow{e_1}, \overrightarrow{e_2}) \le O\left(\frac{\left|C_\ell^{(i)}\right| \eta^3 \log^7 n}{\phi^2}\right)$, *and*

(2) $\left|P_\ell^{(i)}\right| \le O\left(\frac{\eta \log^6 n}{\kappa \phi}\right) \cdot |\boldsymbol{f}|$.

With Lemma A.6.5 (whose proof we defer to Section A.6.2) we can now prove Lemma A.6.4. First we prove the below intermediary lemma. Recall that $\overrightarrow{F}$ is the set of forward edges of F in the residual graph. Let

$$c_{A.6.5} \ge 1 \text{ be the constant hidden in the } O(\cdot) \text{ notation in Lemma A.6.5(1).} \quad \text{(A.6)}$$

Lemma A.6.6. *There are* $g \ge \frac{h}{4}$ *level cuts* $S_{\le i_1}, S_{\le i_2}, \ldots, S_{\le i_g}$ *with* $0 \le i_1 < i_2 < \cdots < i_g \le h$ *such that*

$$\sum_{1 \le j \le g} \boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(S_{\le i_j}, \overline{S_{\le i_j}}) \setminus \overrightarrow{F}\right) \le O\left(|\boldsymbol{f}| \cdot h\right).$$

Proof. Let $P_{\text{all}} \coloneqq \bigcup_\ell \bigcup_i P_\ell^{(i)}$ where the $P_\ell^{(i)}$'s are obtained from Lemma A.6.5. Let

$$\mathcal{S}_{\text{bad}} \coloneqq \left\{0 \le i \le h : E_{G_{\boldsymbol{f}}}(S_{\le i}, \overline{S_{\le i}}) \cap \overrightarrow{\left(\bigcup_\ell X_\ell \setminus P_{\text{all}}\right)} \ne \emptyset\right\}$$

be the set of level cuts that contain at least one expanding edge not in P_{all}, and let $\mathcal{S}_{\text{good}} \coloneqq \{0, 1, \ldots, h\} \setminus \mathcal{S}_{\text{bad}}$. By Lemma A.6.5(1), we know that $|\mathcal{S}_{\text{bad}}| \le c_{A.6.5} \cdot \frac{n\eta^3 \log^7 n}{\phi^2} \cdot \eta \le \frac{h}{4}$ since there are η levels in the hierarchy and our choice of h in (A.4). This means that there are still $|\mathcal{S}_{\text{good}}| = h - |\mathcal{S}_{\text{bad}}| \ge \frac{h}{4}$ "good" level cuts S_i.

By definition of $\operatorname{dist}_{G_{\boldsymbol{f}}}^{\boldsymbol{w}_f}$, an edge e with $\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}(e) > 0$ can be in at most $\boldsymbol{w}_{\boldsymbol{f}}(e)$ level cuts. There are only a few types of edges that can contribute to the size of a good ($i \in \mathcal{S}_{\text{good}}$) level cut $\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}(E_{G_{\boldsymbol{f}}}(S_{\le i}, \overline{S_{\le i}}))$:

(i) Backward edges $\overleftarrow{e}$. These have residual capacities $\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}(\overleftarrow{e}) = \boldsymbol{f}(e)$. The contribution of these (across all good level cuts) can be bounded by

$$\sum_{\overleftarrow{e} \in \overleftarrow{E}} \boldsymbol{c}_{\boldsymbol{f}}^{\kappa}(\overleftarrow{e}) \boldsymbol{w}_G(e) = \sum_{e \in E} \boldsymbol{f}(e) \boldsymbol{w}_G(e) = \boldsymbol{w}_G(\boldsymbol{f}).$$

(ii) Forward edges $\overrightarrow{e}$ from F. These we do not care about in this lemma and will handle later.

(iii) Forward DAG edges $\overrightarrow{e}$. We have set $\boldsymbol{w}(\overrightarrow{e}) = 0$, so they cannot cross a level cut.

(iv) Forward edges $\overrightarrow{e}$, where e is a level-ℓ expanding edge inside some level-ℓ strongly connected component $C_\ell^{(i)}$. By the definition of $\mathcal{S}_{\text{good}}$, we know that $e \in P_\ell^{(i)}$, so there are not too many of these edges. Note that $\boldsymbol{w}_G(\overrightarrow{e}) \le |C_\ell^{(i)}|$ and that these have residual capacity $\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}(e) \le \boldsymbol{c}^{\kappa}(\overrightarrow{e}) = \kappa$ (recall that for the purpose of the analysis, we assume a unit-capacitated multi-graph, i.e., $\boldsymbol{c} = \mathbf{1}$).

As a result, we can bound

$$\begin{aligned}\sum_{i \in \mathcal{S}_{\text{good}}} \boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(S_{\le i}, \overline{S_{\le i}}) \setminus \overrightarrow{F}\right) &\le \boldsymbol{w}_G(\boldsymbol{f}) + \kappa \cdot \left(\sum_{\ell}\sum_{i} \left|P_\ell^{(i)}\right| \cdot \left|C_\ell^{(i)}\right|\right) \\ &\le O\left(|\boldsymbol{f}| \cdot h\right) + \kappa \cdot \eta \cdot O\left(\frac{\eta \log^6 n}{\kappa\phi} \cdot |\boldsymbol{f}| \cdot n\right) \\ &\le O\left(|\boldsymbol{f}| \cdot h\right),\end{aligned}$$

where we used Lemma A.6.5(2) and $\boldsymbol{w}_G(\boldsymbol{f}) = O(|\boldsymbol{f}| \cdot h)$ by Theorem A.4.1(ii). Lemma A.6.6 follows by letting $\{i_1, \ldots, i_g\} \coloneqq \mathcal{S}_{\text{good}}$. □

We can now do a similar ball-growing argument as in Lemma A.5.10 to prove Lemma A.6.4.

Lemma A.6.4. *There exists a level cut $S_{\le i}$ with $0 \le i \le h$ such that*

$$\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}(E_{G_{\boldsymbol{f}}}(S_{\le i}, \overline{S_{\le i}})) \le O(|\boldsymbol{f}|) + \min\{\mathrm{vol}_F(S_{\le i}), \mathrm{vol}_F(\overline{S_{\le i}})\}. \tag{A.5}$$

Proof. Let $S_{\le i_1}, \ldots, S_{\le i_g}$ be the level cuts given by Lemma A.6.6, and let

$$Z \coloneqq \sum_{1 \le j \le g} \boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(S_{\le i_j}, \overline{S_{\le i_j}}) \setminus \overrightarrow{F}\right) \le O(|\boldsymbol{f}| \cdot h).$$

By an averaging argument, at least half of the $S_{\le i_j}$'s satisfy

$$\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(S_{\le i_j}, \overline{S_{\le i_j}}) \setminus \overrightarrow{F}\right) \le \frac{2Z}{g} \le O(|\boldsymbol{f}|). \tag{A.7}$$

Let $i_1^* < \cdots < i_{g/2}^*$ be indices satisfying (A.7), and let $U_j \coloneqq S_{\le i_{j \cdot n}^*} \setminus S_{\le i_{(j-1)\cdot n}^*}$ for each $1 \le j \le \lfloor \frac{g}{2n} \rfloor$. Let $k \coloneqq \lfloor \frac{g}{2n} \rfloor \ge \frac{g}{4n}$. That is, we first split the distance levels into $g/2$ blocks at $i_1^*, \ldots, i_{g/2}^*$, and then merge every n consecutive blocks to form the U_j's. Observe that since $i_{j\cdot n}^* \ge i_{(j-1)\cdot n}^* + n$, we must have

$$\mathrm{dist}_{G_{\boldsymbol{f}}}^{\boldsymbol{w}_F}(U_j, U_{j+2}) > n \tag{A.8}$$

for every j. We will now only consider level cuts that are between some U_j and U_{j+1} and bound the contribution of edges from $\overrightarrow{F}$ to them using a ball-growing argument.

Note that if $\mathrm{vol}_F(U_{\leq 1}) = 0$ or $\mathrm{vol}_F(\overline{U_{\leq k}}) = 0$, then the lemma is vacuously true, and therefore we assume otherwise. We show that there exists a $1 \leq j \leq k$ such that

$$\boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(U_{\leq j}, \overline{U_{\leq j}}) \cap \overrightarrow{F}\right) \leq \min\{\mathrm{vol}_F(U_{\leq j}), \mathrm{vol}_F(\overline{U_{\leq j}})\}, \tag{A.9}$$

which proves the lemma. Assume for contradiction that none of the U_j satisfies (A.9). Because of (A.8) and that the weight of any edge is bounded by n we know that all edges in $E_{G_{\boldsymbol{f}}}(U_{\leq j}, \overline{U_{\leq j}})$ with positive capacities must be in $E_{G_{\boldsymbol{f}}}(U_j, U_{j+1})$. Let $\mathrm{vol}_F^{\kappa}(S) \coloneqq \kappa \mathrm{vol}_F(S)$. If $\mathrm{vol}_F(U_{\leq k/2}) \leq \mathrm{vol}_F(\overline{U_{\leq k/2}})$ then we have

$$\mathrm{vol}_F^{\kappa}(U_{\leq j}) \geq \mathrm{vol}_F^{\kappa}(U_{\leq j-1}) + \boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(U_{\leq j}, \overline{U_{\leq j}}) \cap \overrightarrow{F}\right)$$

for $1 \leq j \leq k/2$ which with the assumption of (A.9) implies that

$$\mathrm{vol}_F^{\kappa}(U_{\leq j}) \geq \left(1 + \frac{1}{\kappa}\right) \mathrm{vol}_F^{\kappa}(U_{\leq j-1}) \implies \mathrm{vol}_F^{\kappa}(U_{\leq k/2}) \geq \left(1 + \frac{1}{\kappa}\right)^{k/2-1} > n^6$$

since $k/2 - 1 \geq k/4 \geq \frac{h}{64n}$ (by Lemma A.6.6 we have $g \geq h/4$) and that $h \geq 1000 n\kappa \log n$ by (A.4). This is a contradiction because the $\mathrm{vol}_F^{\kappa}(S)$ of any S should always be bounded by $2\kappa m \leq n^6$, where recall that $m \leq n^4$ is the total capacities of the input graph and $\kappa \leq n$ is required by Theorem A.6.1. Similarly, if $\mathrm{vol}_F(U_{\leq k/2}) > \mathrm{vol}_F(\overline{U_{\leq k/2}})$, then we have

$$\mathrm{vol}_F^{\kappa}(\overline{U_{\leq j}}) \geq \mathrm{vol}_F^{\kappa}(\overline{U_{\leq j+1}}) + \boldsymbol{c}_{\boldsymbol{f}}^{\kappa}\left(E_{G_{\boldsymbol{f}}}(U_{\leq j}, \overline{U_{\leq j}}) \cap \overrightarrow{F}\right)$$

for $k/2 < j < k$ and thus

$$\mathrm{vol}_F^{\kappa}(\overline{U_{\leq j}}) \geq \left(1 + \frac{1}{\kappa}\right) \mathrm{vol}_F^{\kappa}(\overline{U_{\leq j+1}}) \implies \mathrm{vol}_F^{\kappa}(U_{\leq k/2+1}) \geq \left(1 + \frac{1}{\kappa}\right)^{k/2-1} > n^6.$$

In both cases we have arrived at a contradiction, proving the lemma. □

A.6.2 Robustness of Directed Expander Hierarchy under Flow Augmentation

In this section we prove Lemma A.6.5. There are two main ingredients to this (which are independent of each other), each of which we believe might be of independent interest.

(a) We show a generalization of the classic fact that expanders have low diameters. In particular, in Lemma A.6.7 we show that for any weight function $\boldsymbol{w} \geq \mathbf{0}$, if $\boldsymbol{\nu}$ satisfies $\boldsymbol{\nu}(v) \geq \sum_{e \in \delta_G(v)} \boldsymbol{w}(e)$ and is σ-expanding in G, then the graph has $\boldsymbol{w}$-diameter $\tilde{O}(1/\sigma)$. This indeed generalizes the unweighted pure-expander setting when $\boldsymbol{w} = \mathbf{1}$ and $\boldsymbol{\nu} = \deg_E$.

(b) We show an expander pruning lemma saying that (directed) expanders are robust to path reversals (as well as some other updates, like increasing $\boldsymbol{\nu}$). Indeed, a path reversal changes the size of any (directed) cut by at most one, similar to what happens when deleting an edge. This allows us to show Lemma A.6.9, with similar guarantees as standard expander pruning, but which supports path reversals instead of edge deletions.

In order to prove Lemma A.6.5, for each level-i expander C in $\mathcal{H}$, we set up an appropriate $\boldsymbol{\nu} \in \mathbb{R}^V_{\geq 0}$ and $\sigma \approx 1/n$ such that the fact that $\boldsymbol{\nu}$ is σ-expanding is a certificate that C is initially of low-diameter $\tilde{O}(1/\sigma) = \tilde{O}(n)$ with respect to edge weights $\boldsymbol{w}_G$, via (a). We then show that throughout the run of the push relabel algorithm, a large part of C remains σ-expanding (with respect to $\boldsymbol{\nu}$). Indeed, every time we find an augmenting path in the push relabel algorithm, the residual graph changes by reversing the augmenting path, so we can apply (b). We have to be slightly careful here and use the additional fact that the augmenting paths found by our push relabel algorithm are short (Lemma A.4.8) in order to not blow up the diameter. At the end of the algorithm, (a) will imply that, except for a small pruned part of C, the expanding edges in C remain of low diameter.

Diameter of Expanders with Weighted Edges

We begin by showing (a) in the lemma below, a generalization of the standard fact that expanders have low diameters. Indeed, when $\boldsymbol{\nu}(v) = \deg(v)$ and $\boldsymbol{w}(e) = 1$ it recovers the unweighted case. Note that we are using σ instead of ϕ to avoid confusion with the ϕ in the ϕ-expander hierarchy: One should think of σ as being very small so that $1/\sigma$ corresponds to a certain notion of diameter induced by the weight function $\boldsymbol{w}_G$. In particular, σ can be as small as $\widetilde{O}(1/n)$, while the value ϕ for the expander hierarchy will be set to $1/n^{o(1)}$.

Lemma A.6.7. *Suppose $\boldsymbol{\nu} \in \mathbb{R}^V_{\geq 0}$ is σ-expanding in $H = (V, E)$ and edge weights $\boldsymbol{w} \in \mathbb{N}^E$ such that for all $v \in V$, $\boldsymbol{\nu}(v) \geq \sum_{e \in \delta_H(v)} \boldsymbol{w}(e)$. Then for any $s, t \in V$ such that $\boldsymbol{\nu}(s), \boldsymbol{\nu}(t) > 0$ we have $\operatorname{dist}^{\boldsymbol{w}}_H(s,t) \leq O(\log(\boldsymbol{\nu}(V))/\sigma)$.*

Proof. The proof follows a standard ball-growing argument. Note that $U \coloneqq \{v \in V : \boldsymbol{\nu}(v) > 0\}$ is strongly connected; otherwise, there will be a sparse cut. Let s, t be the vertices with $\boldsymbol{\nu}(s), \boldsymbol{\nu}(t) > 0$ such that $D \coloneqq \operatorname{dist}^{\boldsymbol{w}}_H(s,t)$ is maximized, and assume for contradiction that $D > 16 \left\lceil \frac{\log 4\boldsymbol{\nu}(V)}{\sigma} \right\rceil = O(\log(\boldsymbol{\nu}(V))/\sigma)$. Let $L_i \coloneqq \{v \in U : \operatorname{dist}^{\boldsymbol{w}}_H(s,v) = i\}$.

Let $\boldsymbol{\nu}' \in \mathbb{R}^{\{0,\ldots,D\}}_{\geq 0}$ be defined as follows: First, we add $\boldsymbol{\nu}(L_i)$ to $\boldsymbol{\nu}'(i)$. Then, for each $e \in E$ such that $e = (u,v)$ with $\operatorname{dist}^{\boldsymbol{w}}_H(s,u) < \operatorname{dist}^{\boldsymbol{w}}_H(s,v)$, we add $r_e \coloneqq \frac{\boldsymbol{w}(e)}{\operatorname{dist}^{\boldsymbol{w}}_H(s,v) - \operatorname{dist}^{\boldsymbol{w}}_H(s,u) + 1}$ to $\boldsymbol{\nu}'(i)$ for each $\operatorname{dist}^{\boldsymbol{w}}_H(s,u) \leq i \leq \operatorname{dist}^{\boldsymbol{w}}_H(s,v)$. Observe that $r_e \geq 1/2$ by the fact that $\operatorname{dist}^{\boldsymbol{w}}_H(\cdot,\cdot)$ is the shortest-distance function and $\boldsymbol{w}(e) \geq 1$.

Moreover,

$$\boldsymbol{\nu}(L_{\leq i}) \leq \sum_{0\leq j\leq i} \boldsymbol{\nu}'(j) \leq 2\boldsymbol{\nu}(L_{\leq i}) \quad \text{and} \quad \boldsymbol{\nu}(L_{\geq i}) \leq \sum_{D\geq j\geq i} \boldsymbol{\nu}'(j) \leq 2\boldsymbol{\nu}(L_{\geq i})$$

hold because $\boldsymbol{\nu}(v) \geq \sum_{e\in\delta_H(v)} \boldsymbol{w}(e)$ for all v. By design, for each $0 \leq i < D$ we have

$$\min\{\boldsymbol{\nu}'(i), \boldsymbol{\nu}'(i+1)\} \geq \sum_{e\in E_H(L_{\leq i}, \overline{L_{\leq i}})} r_e \geq \frac{1}{2}\left|E_H(L_{\leq i}, \overline{L_{\leq i}})\right|.$$

Also, by the expansion guarantee of H, we have

$$|E_H(L_{\leq i}, \overline{L_{\leq i}})| \geq \sigma \cdot \min\left\{\boldsymbol{\nu}(L_{\leq i}), \boldsymbol{\nu}(\overline{L_{\leq i}})\right\} \geq \frac{\sigma}{2} \cdot \min\left\{\sum_{j\leq i} \boldsymbol{\nu}'(j), \sum_{j\geq i+1} \boldsymbol{\nu}'(j)\right\}.$$

With these we can now do a standard ball-growing argument. Let $h \coloneqq \lfloor D/2 \rfloor$. If $\sum_{j\leq h} \boldsymbol{\nu}'(j) \leq \sum_{j>h} \boldsymbol{\nu}'(j)$, then

$$\sum_{j\leq i+1} \boldsymbol{\nu}'(j) \geq \left(1 + \frac{\sigma}{4}\right) \cdot \sum_{j\leq i} \boldsymbol{\nu}'(j)$$

holds for all $0 \leq i \leq h$ and therefore

$$\sum_{j\leq h} \boldsymbol{\nu}'(j) \geq \left(1 + \frac{\sigma}{4}\right)^h \cdot \boldsymbol{\nu}'(0) \geq \left(1 + \frac{\sigma}{4}\right)^h \geq 4\boldsymbol{\nu}(V),$$

by $\boldsymbol{\nu}'(0) \geq \boldsymbol{\nu}(s) \geq 1$, which is a contradiction. On the other hand, if $\sum_{j\leq h} \boldsymbol{\nu}'(j) > \sum_{j>h} \boldsymbol{\nu}'(j)$, then similarly

$$\sum_{j\geq i} \boldsymbol{\nu}'(j) \geq \left(1 + \frac{\sigma}{4}\right) \cdot \sum_{j\geq i+1} \boldsymbol{\nu}'(j)$$

holds for all $h \leq i \leq D$ and therefore

$$\sum_{j\geq h} \boldsymbol{\nu}'(j) \geq \left(1 + \frac{\sigma}{4}\right)^{D-h} \cdot \boldsymbol{\nu}'(D) \geq \left(1 + \frac{\sigma}{4}\right)^h \geq 4\boldsymbol{\nu}(V),$$

a contradiction as well. This proves the lemma. □

Expander Pruning under Path-Reversals

Now we show (b) in the lemma below. Note that for our purposes, we only need an existential expander pruning lemma, so we do not care about making it algorithmically efficient. There are a few different types of updates we support, the main ones being reversing a path and adding some volume to $\boldsymbol{\nu}$, tailored for our

use later in this section. We note that the lemma should seamlessly extend to also support edge deletions (as is the usual goal of expander pruning) with the same guarantees as the path reversals, but we do not need it for our purposes, hence we skip it. Since many flow and cut algorithms work via reversing paths, we believe our expander pruning lemma might be of independent interest.

Fact A.6.8. *Let $G = (V, E)$ be a graph and G' be obtained from G by reversing a path in it. Then, we have for each $S \subseteq V$ that $\left||E_G(S, V \setminus S)| - |E_{G'}(S, V \setminus S)|\right| \leq 1$.*

Lemma A.6.9 (Expander Pruning under Path-Reversals)**.** *Given $\boldsymbol{\nu} \in \mathbb{R}^V_{\geq 0}$ such that $\boldsymbol{\nu}$ is σ-expanding in $G = (V, E)$, one can (inefficiently) maintain pruned sets $\emptyset = P^{(0)} \subseteq P^{(1)} \subseteq \cdots \subseteq P^{(k)} \subseteq V$ while G undergoes k updates, the i-th of which either*

(1) adds a vertex v_i to G with volume $\boldsymbol{\nu}(v_i) \coloneqq 0$,

(2) adds an edge e_i to G whose endpoints are not in P,

(3) adds $\Delta_i \in \mathbb{N}$ to $\boldsymbol{\nu}(v_i)$ for some vertex v_i, or

(4) reverses a path R_i in G that does not intersect P,

where P denotes the current pruned set, such that

- *if the i-th update is of type (1) or (2), then[21] $P^{(i)} = P^{(i-1)}$, and*
- *$\boldsymbol{\nu}^{(i)}$ is $\frac{\sigma}{8}$-expanding in $G^{(i)} \setminus P^{(i)}$ and $\boldsymbol{\nu}^{(i)}(P^{(i)}) \leq O(k_i/\sigma + \sum_{j \leq i} \Delta_j)$, with $G^{(i)}$ and $\boldsymbol{\nu}^{(i)}$ denoting the graph and vertex weights after the i-th update and k_i denoting the number of path reversals in the first i updates.*

Proof. We describe how the pruned set $P^{(i)}$ is obtained from $P^{(i-1)}$. We maintain for all i the invariant that $P^{(i)}$ can be written as the disjoint union of two sets $P^{(i)}_+$ and $P^{(i)}_-$ such that[22]

$$\boldsymbol{\nu}^{(i)}(P^{(i)}) \geq \frac{8}{\sigma}\left(\left|E_{G^{(i)}}(P^{(i)}_+, \overline{P^{(i)}})\right| + \left|E_{G^{(i)}}(\overline{P^{(i)}}, P^{(i)}_-)\right| + \left|E_{G^{(i)}}(P^{(i)}_+, P^{(i)}_-)\right|\right). \tag{A.10}$$

Given that $P^{(i-1)}$ satisfies (A.10) in $G^{(i-1)}$, we observe that if we initialize $P^{(i)} \leftarrow P^{(i-1)}$, then it satisfies (A.10) in $G^{(i)}$ since we are not adding edges or reversing paths intersecting P, and increasing vertex weights also only makes the left-hand side larger.

After $P^{(i)}$ is initialized, we repeat the following procedure: As long as there is a cut S that is $\frac{\sigma}{8}$-sparse with respect to $\boldsymbol{\nu}^{(i)}$ in $G^{(i)} \setminus P^{(i)}$, we include it to $P^{(i)}$ by

[21] It is natural that the weight of the prune set is independent from the number of operations (1) and (2), because if $\boldsymbol{\nu}$ was σ-expanding before, then it remains so after such an operation.

[22] Recall that with $E_H(A, \overline{B})$ and $E_H(\overline{B}, A)$, we mean $\overline{B} = H \setminus B$.

setting $P^{(i)} \leftarrow P^{(i)} \cup S$. To see that this does not break the invariant, we assume without loss of generality that S is out-sparse, i.e.,

$$\boldsymbol{\nu}^{(i)}(S) \leq \boldsymbol{\nu}^{(i)}(\overline{S}) \quad \text{and} \quad \left|E_{G^{(i)} \setminus P^{(i)}}(S, \overline{S})\right| < \frac{\sigma}{8} \cdot \boldsymbol{\nu}^{(i)}(S),$$

where $\overline{S} \coloneqq (V(G^{(i)}) \setminus P^{(i)}) \setminus S$. Then, we show that we can add S into $P_+^{(i)}$ while preserving the invariant. To see this, we compute

$$\begin{aligned}
\boldsymbol{\nu}^{(i)}(P^{(i)} \cup S) &= \boldsymbol{\nu}^{(i)}(P^{(i)}) + \boldsymbol{\nu}^{(i)}(S) \\
&\geq \frac{8}{\sigma}\left(\left|E_{G^{(i)}}(P_+^{(i)}, \overline{P^{(i)}})\right| + \left|E_{G^{(i)}}(\overline{P^{(i)}}, P_-^{(i)})\right| + \left|E_{G^{(i)}}(P_+^{(i)}, P_-^{(i)})\right|\right) \\
&\quad + \frac{8}{\sigma}\left|E_{G^{(i)} \setminus P^{(i)}}(S, \overline{S})\right| \\
&\geq \frac{8}{\sigma}\left(\left|E_{G^{(i)}}(P_+^{(i)} \cup S, \overline{P^{(i)} \cup S})\right| + \left|E_{G^{(i)}}(\overline{P^{(i)} \cup S}, P_-^{(i)})\right| + \left|E_{G^{(i)}}(P_+^{(i)} \cup S, P_-^{(i)})\right|\right),
\end{aligned}$$

which is precisely (A.10) for the new $P^{(i)}$ and its partition $(P_+^{(i)}, P_-^{(i)})$. The last inequality follows from that the edges counted in both lines are exactly the same, except for $E_{G^{(i)}}(P_+^{(i)}, S)$ which is counted in the former but not the latter. The case where S is an in-sparse cut can be shown symmetrically (except that it will not be added to $P_-^{(i)}$).

We further show that (A.10) implies the desired upper bound on $\boldsymbol{\nu}^{(i)}(P^{(i)})$. Overloading notation, we extend the vertex set of $V \coloneqq V(G^{(0)})$ to be $V(G^{(i)})$ by adding isolated vertices with weights 0. Note that $\boldsymbol{\nu}^{(0)}$ remains σ-expanding in $G^{(0)}$ after this extension. From this point of view, we see that $G^{(i)}$ is obtained from $G^{(0)}$ by adding edges, increasing vertex weights, and reversing paths; no vertex addition is involved now. We first assume that $\boldsymbol{\nu}^{(0)}(P^{(i)}) \leq 2\boldsymbol{\nu}^{(0)}(V)/3$, i.e., $\boldsymbol{\nu}^{(0)}(P^{(i)}) \leq 2\min\{\boldsymbol{\nu}^{(0)}(P^{(i)}), \boldsymbol{\nu}^{(0)}(\overline{P^{(i)}})\}$. We know by the expansion guarantee of $G^{(0)}$ that both $P_+^{(i)}$ and $P_-^{(i)}$ are not sparse in $G^{(0)}$ and therefore

$$\begin{aligned}
\boldsymbol{\nu}^{(0)}(P^{(i)}) = \boldsymbol{\nu}^{(0)}(P_+^{(i)}) + \boldsymbol{\nu}^{(0)}(P_-^{(i)}) &\leq \frac{2}{\sigma}\left(\left|E_{G^{(0)}}(P_+^{(i)}, \overline{P_+^{(i)}})\right| + \left|E_{G^{(0)}}(\overline{P_-^{(i)}}, P_-^{(i)})\right|\right) \\
&= \frac{2}{\sigma}\left(\left|E_{G^{(0)}}(P_+^{(i)}, \overline{P^{(i)}})\right| + \left|E_{G^{(0)}}(\overline{P^{(i)}}, P_-^{(i)})\right| + 2\left|E_{G^{(0)}}(P_+^{(i)}, P_-^{(i)})\right|\right),
\end{aligned}$$

where the first inequality was based on

$$\min\left\{\boldsymbol{\nu}^{(0)}(P_+^{(i)}), \boldsymbol{\nu}^{(0)}(\overline{P_+^{(i)}})\right\} \geq \frac{1}{2}\boldsymbol{\nu}^{(0)}(P_+^{(i)})$$

and

$$\min\left\{\boldsymbol{\nu}^{(0)}(P_-^{(i)}), \boldsymbol{\nu}^{(0)}(\overline{P_-^{(i)}})\right\} \geq \frac{1}{2}\boldsymbol{\nu}^{(0)}(P_-^{(i)})$$

by our assumption. As such, we have

$$\begin{aligned}
\boldsymbol{\nu}^{(i)}(P^{(i)}) &\le \frac{2}{\sigma}\left(\left|E_{G^{(0)}}(P_+^{(i)},\overline{P^{(i)}})\right| + \left|E_{G^{(0)}}(\overline{P^{(i)}},P_-^{(i)})\right| + 2\left|E_{G^{(0)}}(P_+^{(i)},P_-^{(i)})\right|\right) + \sum_{j\le i}\Delta_j \\
&\overset{(i)}{\le} \frac{2}{\sigma}\left(\left|E_{G^{(i)}}(P_+^{(i)},\overline{P^{(i)}})\right| + \left|E_{G^{(i)}}(\overline{P^{(i)}},P_-^{(i)})\right| + 2\left|E_{G^{(i)}}(P_+^{(i)},P_-^{(i)})\right| + 2k_i\right) + \sum_{j\le i}\Delta_j \\
&\le \frac{2}{\sigma}\left(2\left|E_{G^{(i)}}(P_+^{(i)},\overline{P^{(i)}})\right| + 2\left|E_{G^{(i)}}(\overline{P^{(i)}},P_-^{(i)})\right| + 2\left|E_{G^{(i)}}(P_+^{(i)},P_-^{(i)})\right|\right) + \frac{4k_i}{\sigma} + \sum_{j\le i}\Delta_j \\
&\overset{(ii)}{\le} \frac{\boldsymbol{\nu}^{(i)}(P^{(i)})}{2} + \frac{4k_i}{\sigma} + \sum_{j\le i}\Delta_j,
\end{aligned}$$

where (i) follows from

$$E_{G^{(r)}}(P_+^{(i)},\overline{P^{(i)}}) \cup E_{G^{(r)}}(P_+^{(i)},P_-^{(i)}) = E_{G^{(r)}}(P_+^{(i)},\overline{P_+^{(i)}})$$

and

$$E_{G^{(r)}}(\overline{P^{(i)}},P_-^{(i)}) \cup E_{G^{(r)}}(P_+^{(i)},P_-^{(i)}) = E_{G^{(r)}}(\overline{P_-^{(i)}},P_-^{(i)})$$

for $r \in \{0,i\}$ with Fact A.6.8, and (ii) follows from (A.10). This implies $\boldsymbol{\nu}^{(i)}(P^{(i)}) \le O(k_i/\sigma + \sum_{j\le i}\Delta_j)$.

The case when $\boldsymbol{\nu}^{(0)}(P^{(i)}) > 2\boldsymbol{\nu}^{(0)}(V)/3$ can be argued similarly: Consider the moment when we added S to $P^{(j)}$ for some $j \le i$ such that $\boldsymbol{\nu}^{(0)}(P^{(j)}) \le 2\boldsymbol{\nu}^{(0)}(V)/3$ but $\boldsymbol{\nu}^{(0)}(P^{(j)} \cup S) > 2\boldsymbol{\nu}^{(0)}(V)/3$. Applying the calculation above we know that $\boldsymbol{\nu}^{(j)}(P^{(j)}) \le O(k_j/\sigma + \sum_{k\le j}\Delta_j)$ at this moment, before S is included. This gives that

$$\begin{aligned}
\frac{2}{3}\boldsymbol{\nu}^{(0)}(V) &< \boldsymbol{\nu}^{(0)}(P^{(j)}) + \boldsymbol{\nu}^{(0)}(S) \\
&\le \boldsymbol{\nu}^{(j)}(P^{(j)}) + \boldsymbol{\nu}^{(j)}(S) \\
&\le O(k_j/\sigma + \sum_{k\le j}\Delta_k) + \frac{\boldsymbol{\nu}^{(j)}(V)}{2} \\
&\le O(k_j/\sigma + \sum_{k\le j}\Delta_k) + \frac{\boldsymbol{\nu}^{(0)}(V)}{2} + \sum_{k\le j}\Delta_k,
\end{aligned}$$

which implies $\boldsymbol{\nu}^{(0)}(V) \le O(k_j/\sigma + \sum_{k\le j}\Delta_k) \le O(k_i/\sigma + \sum_{j\le i}\Delta_j)$. Since $\boldsymbol{\nu}^{(i)}(P^{(i)})$ can be trivially upper bounded by $\boldsymbol{\nu}^{(i)}(V)$, we get $\boldsymbol{\nu}^{(i)}(P^{(i)}) \le \boldsymbol{\nu}^{(0)}(V) + \sum_{j\le i}\Delta_j \le O(k_i/\sigma + \sum_{j\le i}\Delta_j)$, which is the bound we wanted. This completes the proof of Lemma A.6.9. □

Proof of Lemma A.6.5

Now we have the two ingredients (a) and (b) in order to prove our Lemma A.6.5.

Lemma A.6.5. *There exists a subset $P_\ell^{(i)} \subseteq X_\ell^{(i)}$ such that*

(1) for each pair $e_1, e_2 \in X_\ell^{(i)} \setminus P_\ell^{(i)}$, we have $\mathrm{dist}_{G_{\boldsymbol{f}}}^{\boldsymbol{w_f}}(\overrightarrow{e_1}, \overrightarrow{e_2}) \leq O\left(\frac{\left|C_\ell^{(i)}\right| \eta^3 \log^7 n}{\phi^2}\right)$, and

(2) $\left|P_\ell^{(i)}\right| \leq O\left(\frac{\eta \log^6 n}{\kappa\phi}\right) \cdot |\boldsymbol{f}|$.

Let us focus on a level-ℓ expander C. We first describe the high-level strategy of the proof.

Constructing Initial Expander. By our analysis in Section A.5, we know that all expanding edges in C are only $\tilde{O}(|C|/\phi)$ far away from each other, with respect to $\boldsymbol{w}_G$-distance. This lets us find an appropriate $\boldsymbol{\nu} \in \mathbb{R}^H_{\geq 0}$ and $\sigma \approx \frac{\phi^2}{|C|}$, in Claim A.6.11, such that $\boldsymbol{\nu}(v) \geq \sum_{e \in \delta_H(v)} \boldsymbol{w}_G(e)$ and $\boldsymbol{\nu}$ is σ-expanding in H, where H is some subgraph of C^κ.[23] By our generalized "expanders have low diameter" argument in Lemma A.6.7, $(H, \boldsymbol{\nu}, \sigma)$ is now a certificate/witness that the expanding edges in H are of low-diameter $\tilde{O}(|C|/\phi^2)$. We will set H to the graph we would get if we run a cut-matching game on C, where in each round we find a *short* matching. The graph H will precisely consist of edges certifying that C is of low diameter, but not include irrelevant parts of C which might be far from all expanding edges in C.

Handling Path-Reversal. Next we will consider how the graph develops when we run our push-relabel augmenting paths algorithm. Throughout, we will maintain $(H, \boldsymbol{\nu}, \sigma)$ and a small pruned set P as a certificate/witness that most edges from C are still of low-diameter, via our expander pruning Lemma A.6.9. In particular, $\boldsymbol{\nu}$ will be $\Theta(\sigma)$-expanding in $H \setminus P$.

(a) Truncating the Path. In particular, consider what happens when we want to reverse an augmenting path R. Let R' be the subpath from the first vertex in $H \setminus P$ to the last vertex in $H \setminus P$. Note that when focusing on H, we do not care about how the path R looks like outside of the subpath R'. Nevertheless, note that it is still possible for R' to go outside of $H \setminus P$ (to $V \setminus H$ or P).

(b) Bounding Path Length. We first notice that, since our push relabel algorithm almost finds shortest paths (Lemma A.4.8), it must be the case that R' is of $\boldsymbol{w}_G$-length $\tilde{O}(|C|/\phi^2)$ since $H \setminus P$ is still of low diameter.

(c) Adding New Vertices to H. We add all vertices on R' which are not already in $H \setminus P$ as "fresh" vertices to H (in particular, if R' intersects the pruned set P we add back new copies of these vertices), and add all the edges of R' not already in $H \setminus P$ to H (using operations (1) and (2) in Lemma A.6.9). Note

[23] Recall that C^κ is the graph with all edges duplicated κ times—indeed, the flow algorithm will work in this graph.

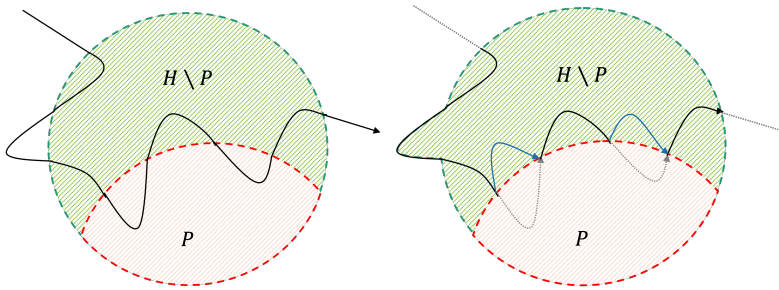


Figure A.2: Illustration of how a path reversal is handled. After truncating the path from the first intersection with $H \setminus P$ to the last, we add vertices on the path not in H into the graph. If the path goes into the pruned set P, we also add the corresponding "fresh" vertices to the graph and "reroute" the path segment inside the un-pruned part which creates the blue segments. The final path that we reverse (via Lemma A.6.9(4)) at the end consists of the black and blue path segments.

that H might no longer be a subgraph of C (as R' can move outside of C). In fact, this is necessary: the expanding edges of C will at the end of the push relabel algorithm be of low diameter inside of $G^{\kappa}_{\boldsymbol{f}}$, but not necessarily inside the induced subgraph $G^{\kappa}_{\boldsymbol{f}}[C]$. We remark that technically H may now contain multiple copies of the same vertex in G, but only one of these copies will be "active" and the others will be in P.

(d) Performing Path-Reversal. We then reverse the path R' (using operation (4) in Lemma A.6.9), which might increase the pruned set P a bit.

(e) Increasing Vertex Weights. Additionally we must increase $\boldsymbol{\nu}(v)$ for the vertices v incident to R' a bit, to maintain that $\boldsymbol{\nu}(v) \geq \sum_{e \in \delta_H(v)} \boldsymbol{w}_G(e)$ after we added some edges to H (using operation (3) in Lemma A.6.9). This again might increase the pruned set P a bit, and thus we will increase $\boldsymbol{\nu}(v)$ proportional to the $\boldsymbol{w}_G$-length of R' (which we argued above is not too long) in order to control this blow-up.

At the end, after all augmenting paths, $\boldsymbol{\nu}$ is still $\Theta(\sigma)$-expanding in $H \setminus P$, and the pruned set P is small, which lets us conclude Lemma A.6.5.

We begin with this useful claim, which will allow us to set up the appropriate vertex volume $\boldsymbol{\nu}$.

Claim A.6.10. *For every* 1*-respecting demand* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ *on* $X^{(i)}_{\ell}$ *we can route it by an integral flow in* $C^{(i)}_{\ell}$ *with congestion* $O(\eta \log n/\phi)$ *such that each flow path has* $\boldsymbol{w}_G$*-length at most* $O\left(\left|C^{(i)}_{\ell}\right| \eta^2 \log n/\phi\right)$.

Proof. Every 1-respecting demand on $X^{(i)}_{\ell}$, by definition of expansion and the max-flow min-cut theorem Fact A.3.4, is routable in $C^{(i)}_{\ell}$ with congestion $\frac{1}{\phi}$, and hence

also with integral congestion $\lceil \frac{1}{\phi} \rceil \leq \frac{2}{\phi} = O(\frac{1}{\phi})$ (since $\phi \leq 1$). Note that we may restrict the expander hierarchy $\mathcal{H}$ to $C_\ell^{(i)}$ (only keeping expanding edges of level at most ℓ). The claim then follows from Corollary A.5.16 (we do not use edges from F for this routing, so we have $\boldsymbol{w}_G(e) = \boldsymbol{w}_{\mathcal{H}}(e)$). □

Proof of Lemma A.6.5. Let $X \coloneqq X_\ell^{(i)}$ and $C \coloneqq C_\ell^{(i)}$ (that is, X are the expanding edges inside some level-ℓ expander C). Consider running the cut-matching game[24] of Theorem A.3.6 on $\deg_X$ to construct a witness W (embeddable into C) in which $\deg_X$ is ψ_{CMG}-expanding (with $\frac{1}{\psi_{\text{CMG}}} = O(\log^2 n)$). Every time we are given a bipartition $(\boldsymbol{\nu}_A, \boldsymbol{\nu}_B)$ of $\deg_X$, we apply Claim A.6.10 to find a matching $\overrightarrow{M}$ and $\overleftarrow{M}$ between vol_A and vol_B that are routable in C with congestion $O(\eta \log n/\phi)$ such that each edge is embedded into a path of $\boldsymbol{w}_G$-weight $O\left(|C|\,\eta^2 \log n/\phi\right)$. Overall, after $t_{\text{CMG}} = O(\log^2 n)$ rounds, we get a witness W embeddable into C with congestion $\kappa_W \coloneqq O(\eta \log^3 n/\phi)$ where each edge of W is embedded into a path of $\boldsymbol{w}_G$-length $D \coloneqq O\left(|C|\,\eta^2 \log n/\phi\right)$. We let $H_0 \subseteq C$ be the image of the embedding (that is, H_0 consist of the union (after removing duplicates) of edges on all paths in the embedding of W to C, and only vertices incident to those edges).

We will construct vertex volumes $\boldsymbol{\nu}_0 \in \mathbb{N}^{V(H_0)}$ as follows. For each edge $e_W = (u, v) \in E(W)$, let $P_{e_W} \subseteq H_0$ be the embedding path of (u, v) into H_0. For each $e \in P_{e_W}$, we add $\boldsymbol{w}_G(e)$ to the vertex weights of both of its endpoints. We then add D to both $\boldsymbol{\nu}_0(u)$ and $\boldsymbol{\nu}_0(v)$. (Now, note that $\boldsymbol{\nu}_0$ is almost a scaled up version of $\deg_X$: in fact $\boldsymbol{\nu}_0 = \boldsymbol{\nu}_0^a + \boldsymbol{\nu}_0^b$ where $D \cdot \deg_X \leq \boldsymbol{\nu}_0^a \leq t_{\text{CMG}} D \cdot \deg_X$, and $\|\boldsymbol{\nu}_0^b\|_1 \leq 2\|\boldsymbol{\nu}_0^a\|_1$). This construction guarantees $\boldsymbol{\nu}_0(v) \geq \sum_{e \in \delta_{H_0}(v)} \boldsymbol{w}(e)$.

Since $\deg_X$ is ψ_{CMG}-expanding in W and embeddable into H_0 with congestion κ_W, and as we noted before, $\boldsymbol{\nu}_0 \approx D \cdot \deg_X$, the following claim is reasonable.

Claim A.6.11. *If $\sigma \coloneqq \frac{\psi_{CMG}}{4D\kappa_W} = \Omega\left(\frac{1}{\kappa_W D \log^2 n}\right)$, then $\boldsymbol{\nu}_0$ is σ-expanding in H_0.*

Proof. Consider some cut $S \subseteq V(H_0)$, with $\boldsymbol{\nu}_0(S) \leq \boldsymbol{\nu}_0(\overline{S})$. We want to argue that $|E_{H_0}(S, \overline{S})| \geq \sigma \boldsymbol{\nu}_0(S)$ and $|E_{H_0}(\overline{S}, S)| \geq \sigma \boldsymbol{\nu}_0(S)$. We argue the former, and the latter is symmetric.

Consider the multiset of edges $E' = \bigcup_{e_W \in E(W)} P_{e_W}$ from the embedding from W to H_0. Since this embedding has congestion κ_W, we know that $|E'(S, \overline{S})| \leq \kappa_W |E_{H_0}(S, \overline{S})|$, so it suffices to show that $|E'(S, \overline{S})| \geq \frac{\psi_{\text{CMG}}}{4D} \boldsymbol{\nu}_0(S)$. We write $E'(S, \overline{S}) = E'_1 \cup E'_2$, where E'_1 consists of those edges e which comes from embedding paths P_{e_W} where $e_W \in E_W(S, \overline{S})$, and E'_2 are the remaining ones.

We now bound $\boldsymbol{\nu}_0(S)$ as follows. Recall that for each edge $e_W = (u, v) \in E(W)$, we added D to $\boldsymbol{\nu}_0(u)$ and $\boldsymbol{\nu}_0(v)$, as well as $\boldsymbol{w}_G(e)$ to the two endpoints of e for each e on P_{e_W}. In particular, the total contribution of e_W to all of $\boldsymbol{\nu}_0$ (and thus to

[24] While the cut-matching game in Theorem A.3.6 is randomized and only works with high probability, here we may simply assume that the randomness used in the cut-matching game are such that it succeeds (indeed such random bits exists, and in this section we only need *existence* of the following witness W and embedding).

$\boldsymbol{\nu}_0(S)$) is at most $4D$. If either u or v is in S, we can charge this cost of $4D$ to the contribution of e_W to $\mathrm{vol}_{E(W)}(S)$.

The only volume in $\boldsymbol{\nu}_0(S)$ we have not accounted for now, is exactly the volume coming from edges $e_W = (u, v) \in E(W)$ where $u, v \in \overline{S}$, but for which the path P_{e_W} intersects S. Such edges e_W can contribute at most $2D$ to the volume of $\boldsymbol{\nu}_0(S)$, and they must also contribute at least one edge in E_2'.

The above reasoning shows that $\boldsymbol{\nu}_0(S) \leq 2D \cdot |E_2'| + 4D \cdot \mathrm{vol}_{E(W)}(S)$. Since W is a ψ_{CMG}-expander, we have $|E_1'| \geq |E_W(S, \overline{S})| \geq \psi_{\mathrm{CMG}} \mathrm{vol}_{E(W)}(S)$ (each edge $e_W \in E_W(S, \overline{S})$ must clearly have a counterpart in E_1'). Thus we conclude $\boldsymbol{\nu}_0(S) \leq 2D \cdot |E_2'| + 4D \cdot |E_1'|/\psi_{\mathrm{CMG}}$, and hence that $\frac{\psi_{\mathrm{CMG}}}{4D} \boldsymbol{\nu}_0(S) \leq |E_1'| + |E_2'|$, which proves the lemma. □

Setup. Now we can initialize $\boldsymbol{\nu} \leftarrow \kappa \cdot \boldsymbol{\nu}_0$ and $H \leftarrow (H_0)^\kappa$ (that is H_0, but with each edge duplicated κ times). By construction, H is a subgraph of G^κ, and from Claim A.6.11 we know that $\boldsymbol{\nu}$ is σ-expanding in H. Also, by construction, $\boldsymbol{\nu}(v) \geq \sum_{e \in \delta_H(v)} \boldsymbol{w}(e)$ (and hence by Lemma A.6.7, of diameter $O(\frac{\log(n\kappa)}{\sigma})$, and recall that $\kappa \leq \mathrm{poly}(n)$). We also know that $\boldsymbol{\nu} \geq \kappa \cdot D \cdot \deg_X$.

Now we consider reversing each flow path in $\boldsymbol{f}$ one at a time, following the order the paths are discovered by the push-relabel algorithm. For simplicity we regard $\boldsymbol{f}$ as a flow in the subdivided graph G^κ, and hence that each flow path sends exactly a single unit of flow. We are going to maintain the subgraph H of $G^\kappa_{\boldsymbol{f}}$ and that $\boldsymbol{\nu}$ is $\frac{\sigma}{8}$-expanding in $H \setminus P$ for a pruned set P throughout the reversals via Lemma A.6.9. In fact, we will sometimes need to add back vertices from P into H, and when we do so we will add them as fresh/forked new vertices. Therefore technically speaking H will not necessarily be a subgraph of $G^\kappa_{\boldsymbol{f}}$, since some vertices might occur more than once in H. However, all but one copy of each vertex will be in P, so we still always maintain that $H \setminus P$ is subgraph of $G^\kappa_{\boldsymbol{f}}$.

Low-Diameter Invariant. After each path reversal, we will increase some of the vertex weights via Lemma A.6.9 to ensure that $\boldsymbol{\nu}(v) \geq \sum_{e \in E(H \setminus P) \cap \delta(v)} \boldsymbol{w}_G(e)$ holds for all $v \in V(H) \setminus P$. Note that by construction of H and $\boldsymbol{\nu}$, this holds initially. This together with the fact that $\boldsymbol{\nu}$ is $\frac{\sigma}{8}$-expanding in $H \setminus P$ shows that the subgraph $G[V(H) \setminus P]$ has $\boldsymbol{w}_G$-diameter $O(\log n/\sigma)$ by Lemma A.6.7.

Dealing with Path Reversal. Suppose we have dealt with the first $j-1$ flow paths already, and now we are preparing to reverse the j-th flow path R_j in $G^\kappa_{\boldsymbol{f}_{j-1}}$. If R_j does not intersect with $V(H)$, then nothing needs to be done. Otherwise, we take the first point s_j and the last point t_j on R_j that intersect $V(H) \setminus P$ and replace R_j with the subpath between them. We now ensure to add all vertices from R_j to H, which are not already in $H \setminus P$ via Lemma A.6.9. Importantly, when we add an already pruned vertex $v \in P$, we use a fresh instance of this vertex (so that the newly added v' will be in $V(H)$ but not in P, see also Figure A.2).

Indeed Lemma A.6.9 only allows reversing paths that do not intersect P. Since $s_j, t_j \in V(H) \setminus P$, by the low-diameter invariant above we know that

$$\mathrm{dist}^{\boldsymbol{w}_G}_{G^\kappa_{\boldsymbol{f}_{j-1}}}(s_j, t_j) = O(\log n/\sigma)$$

and thus $\boldsymbol{w}_G(R_j) = O(\log n/\sigma)$ as well by Lemma A.4.8 (recall $\boldsymbol{f}$ is obtained by running our push-relabel algorithm of Theorem A.4.1 on G^κ, and the push relabel algorithm will find almost shortest paths). We can now for each $e \in R_j$ add e to H. After doing so, we go back and for each of the new edge (u, v) added to H, we increase $\boldsymbol{\nu}(u)$ and $\boldsymbol{\nu}(v)$ by $\boldsymbol{w}_G(e)$ to ensure that $\boldsymbol{\nu}(v) \geq \sum_{e \in E(H \setminus P) \cap \delta(v)} \boldsymbol{w}_G(e)$ holds for all $v \in V(H) \setminus P$. The reason why we first add all edges and then do the vertex weight increments is to make sure the pruned set does not grow while adding edges. Note that these edge additions are valid as none of them are incident to P due to us using fresh vertices. By $\boldsymbol{w}_G(R_j) = O(\log n/\sigma)$, we also conclude that the total amount we just added to the vertex weights is $O(\log n/\sigma)$.

Finally, we reverse R_j in H, via Lemma A.6.9. The pruned set P may grow according to Lemma A.6.9 after each vertex volume increment and path reversal. One can verify that all the invariants are maintained.

Summary. In total, we have $|\boldsymbol{f}|$ path reversals, and each also increased $\|\boldsymbol{\nu}\|_1$ by $O(\log n/\sigma)$. At the end, by Lemma A.6.9, the total volume of the pruned set P is bounded by:

$$\begin{aligned}\boldsymbol{\nu}(P) &= O\left(|\boldsymbol{f}|\left(\frac{1}{\sigma} + \frac{\log n}{\sigma}\right)\right)\\ &= O\left(|\boldsymbol{f}| \cdot \kappa_W D \log^3 n\right)\\ &= O\left(|\boldsymbol{f}| \cdot D \cdot \frac{\eta \log^6 n}{\phi}\right)\end{aligned}$$

Since we have $\boldsymbol{\nu} \geq \kappa \cdot D \cdot \deg_X$ initially (and this never changes as $\boldsymbol{\nu}$ only grows), $\mathrm{vol}_X(P) \leq O(\frac{|\boldsymbol{f}|\eta \log^6 n}{\kappa\phi})$, and hence P can be incident to at most $O(\frac{|\boldsymbol{f}|\eta \log^6 n}{\kappa\phi})$ many edges from X. The rest of X, by the low-diameter invariant, are reachable from each other in H (and hence in $G^\kappa_{\boldsymbol{f}}$) by a path of $\boldsymbol{w}_G$-length $O(\log n/\sigma) = O\left(\frac{|C|\eta^3 \log^7 n}{\phi^2}\right)$. This completes the proof of Lemma A.6.5. □

A.7 Building an Expander Hierarchy

In this section, we show how to construct an expander hierarchy of the input graph that was used earlier in this paper for deriving the weight function needed by our push-relabel algorithm in Section A.5.

Theorem A.7.1. *There is a randomized algorithm that, given an n-vertex capacitated simple graph $(G, \boldsymbol{c})$, with high probability constructs a $1/n^{o(1)}$-expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of $(G, \boldsymbol{c})$ with $\eta = O(\log n)$ in $n^{2+o(1)}$ time.*

In particular, we show the following Theorem A.7.2, from which Theorem A.7.1 immediately follows if we choose, e.g., $\phi = \exp\left(-\frac{\log n}{(\log\log n)^{1/3}}\right)$.

Theorem A.7.2. *Given an n-vertex simple capacitated graph $(G, \boldsymbol{c})$ and a parameter $0 < \phi < 2^{-\omega\left(\frac{\log n}{\sqrt{\log\log n}}\right)}$ sufficiently small, there is a randomized $\frac{n^{2+o(1)}}{\phi^3}$ time algorithm that with high probability constructs a $\phi/n^{o(1)}$-expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of $(G, \boldsymbol{c})$ with $\eta = O(\log n)$.*

In fact, our Theorems A.7.1 and A.7.2 achieve an additional property that each X_i is a separator of $G \setminus X_{>i}$, where X is a *separator* of H if none of the edges in F has both endpoints in the same strongly connected component of H. This is because we will construct each X_i by repeatedly finding cuts in G and removing edges from one of the directions (i.e., $E_G(S, \overline{S})$ or $E_G(\overline{S}, S)$ for some S) which disconnects the two sides of the cuts.

A.7.1 Overview and Setup

We first give a high-level overview of the algorithm, where for simplicity we assume the graph is unit-capacitated, since both the analysis and the algorithm itself extend seamlessly to the capacitated setting. Note that the first level of the expander hierarchy is easy to compute via standard expander decomposition techniques. Recall that $G_i \coloneqq G \setminus X_{>i}$ for any i. That is, we can get three edge sets D, X_1, X_2 such that D is a DAG, X_1 is ϕ-expanding in G_1, and $|X_2|$ is small (on the order of ϕm). To construct the second level and onward, one immediate idea is to simply do expander decomposition with respect to the *volume induced by* X_2, which is in fact doable by incorporating our sparse-cut algorithm of Theorem A.6.1 into the framework of e.g., [NSW17; BPS20; HKPW23]. If the returned edge set X_3 happens to be a subset of X_2, then we can set $X_2 \leftarrow X_2 \setminus X_3$ and continue to run expander decomposition on X_3. As the number of edges in the terminal set decreases roughly by a factor of ϕ each time, after $O(\log_{1/\phi} n)$ iterations we will get the desired expander hierarchy.

The issue is that the edge set X_3 we need to cut when doing expander decomposition with respect to the volume induced by X_2 may not be a subset of X_2 and hence, it "cuts through" the strong components of G_1. Indeed, it might necessarily be the case that X_3 includes edges from X_1 or even from D; in general given any $F \subseteq E$ there might not be a separator contained in F that makes F expanding in the remaining graph. Having $X_3 \subsetneq X_2$ would result in a *non-nested* expander hierarchy which is incompatible with our sparse-cut algorithm once we recurse on both sides of the cut.

To further understand why this breaks the previous layers, notice that as $X_3 \subsetneq X_2$ the graph $G_1 = G \setminus X_{>1}$ in which X_1 is expanding changes. This would potentially decrease the well-connectivity of G_1 and make X_1 no longer expanding. To overcome this, we apply a seemingly naïve approach: Whenever we find X_3, we immediately

remove $X_3 \setminus X_2$ from G_1 (note that it suffices to remove $X_3 \setminus X_2$). We then try to further refine the strongly connected components of G_1 into smaller pieces so that X_1 is still expanding in this graph. As a result, some edges that were previously in X_1 got removed from G_1, and to accommodate them we further add these edges into X_2. Since the volume induced by X_2 increases, it may no longer be expanding in $G_2 = G \setminus X_3$, and to fix it we similarly refine G_2 by putting more edges into X_3, which in turn may result in us moving even more volume to X_2, and so on. While this creates a loop between the two steps that seemingly takes $\Omega(n)$ rounds, we show that with careful analysis and algorithmic implementation, this number can actually be bounded.

Intuition of Analysis

Ideal Scenario. To see why the number of iterations can be bounded, let us first consider the ideal case that (1) if we remove D edges from G_1, then we can find a set D' of $O(D)$ edges in G_1 to be further removed so that X_1 remains expanding in it, and (2) if we add A new edges to X_2, then we can find a set A' of $O(\phi A)$ edges to be removed from G_2 (hence added to X_3) so that X_2 remains expanding in G_2. In this case, we can easily see that the number of edges to be removed from G_1 and the number of edges to be added to X_2 in fact decrease by an $O(\phi) < 1/2$ factor each round, which means that the number of rounds is bounded by $O(\log n)$. If we further consider the third level X_3, then we can see that there are $O(\log n)$ rounds of interaction between X_2 and X_3, each of which generates another $O(\log n)$ rounds of interaction between X_1 and X_2. Therefore, we can bound the total number of iterations of the algorithm by $O(\log n)^\eta$ where η is the height of the final hierarchy we construct. To this end, notice that the number of terminal edges is reduced by roughly a factor of $O(\phi)$ in each level, and thus we can bound η by roughly $O(\log_{1/\phi} m)$. Choosing $\phi < 1/n^{o(1)}$ sufficiently small (for instance, $\phi = 2^{-\sqrt{\log n}}$), this shows that the algorithm will terminate in $O(\log n)^{O(\log_{1/\phi} m)} = n^{o(1)}$ iterations, which is what we are aiming for.

The question thus now becomes: Is this ideal scenario achievable? Existentially, by standard expander arguments, such edge sets always exist. Thus, we may hope to generalize and apply existing expander pruning algorithms (e.g., [SW19; BPS20; HKPW23; SP24]) to locate them by replacing the flow algorithm used by these frameworks with the sparse-cut subroutine we developed in Section A.6. Note that it is NP-hard to compute expander decomposition/pruning exactly, but as in most standard approaches we can afford some multiplicative approximation as long as the number of edges still goes down by half each round.

Fixing Hierarchy with Few Cuts. However, none of these algorithms locate the entire edge sets in one shot. Instead, they work by repeatedly finding sparse cuts in the graph and recurse on both sides U_1 and U_2 of the cut. But notice that our Theorem A.6.1 when running on subgraph $G[U]$ requires an expander hierarchy

of $G[U] \setminus X_i$ (when we are at level i trying to build expander decomposition with respect to X_i). Although at the beginning of the algorithm, we have obtained from the previous layers a hierarchy $\mathcal{H}_{\text{prev}}$ of $G \setminus X_i$, if the first sparse cut we found is not contained in X_i, then we can no longer extract a valid hierarchy of $G[U] \setminus X_i$ from $\mathcal{H}_{\text{prev}}$. In this case, we need to first go down to the pervious layers and fix their hierarchy before coming back to level i and continue locating sparse cuts. This invalidates our previous ideal analysis.

Fortunately for us, some of these previous algorithms (specifically [BPS20; HKPW23]) follow the framework established by [NS17; Wul17] which allows one to argue that we can locate all these edges in n^{ε} calls for some $\varepsilon = o(1)$ to the sparse-cut subroutines in total.[25] As a result, the number of times we need to go back to the previous level is bounded by roughly n^{ε} (the $O(\log n)$ factor induced by the reduction of edges is overwhelmed by this term). Choosing ϕ to be even smaller (yet still $1/n^{o(1)}$), we can ensure that the total of calls to the sparse-cut subroutines throughout the whole construction is $n^{\varepsilon \log_{1/\phi} n} = n^{o(1)}$.

Amortized vs Expected Worst-Case Recourse. Another issue with applying previous approaches is that these algorithms only have *amortized* recourse guarantees. For example, in the first case where we remove D edges from G_1, instead of always returning an edge set D' of $O(D)$ edges, the amortized guarantee only ensures that if we remove k batches $D_1, \ldots, D_k$ edges from G_1, then the algorithm returns $D'_1, \ldots, D'_k$ such that $|D'_1| + \cdots + |D'_i| = O(D_1 + \cdots + D_i)$ for every $i \in [k]$. Unfortunately, amortized guarantees would break the above analysis of having the size of the edge sets reduced by half each iteration, as we might have a single large update early, and then a large number of small updates later with no decrease in size.

To overcome this, we observe that while worst-case output recourse might be algorithmically challenging to achieve in these algorithms, our analysis works if we can obtain a weaker *expected* worst-case recourse. Indeed, consider again the ideal scenario except that the size of D' and A' is only $O(D)$ and $O(\phi A)$ *in expectation* respectively. By the law of total expectation, we can argue that the expected number of edges needed to be fixed still decreases by half each iteration. Even though we can no longer conclude that the number of iterations is bounded by exactly $\log m$, notice that after $100 \log m$ rounds the expected number of edges needed to be fixed drops to at most m^{-99}. By Markov's inequality, this still shows that with high probability the interaction between the two levels is bounded by $100 \log m$. In Section A.7.1 we describe how we modify previous algorithms to achieve the expected worst-case output recourse.

We remark that this is the only place in our analysis that requires randomness and the only reason why our final algorithm is not deterministic—indeed, the

[25] This is not technically accurate as we do still need to recurse on the smaller side of the cuts, but in this case we get a size reduction and all the recursions are vertex-disjoint.

randomized cut-matching game of [Lou10] can be easily replaced with a deterministic one [BPS20].

A Data Structure Point-of-View.

To formally capture the interaction between the current level and the previous levels, we employ a data structure perspective and define the following. In the remainder of the section, let $m \leq n^4$ be the total capacities of the edges in G.

Definition A.7.3. A $(k, \alpha, \beta, \phi, T)$*-hierarchy maintainer* $\mathcal{M}$ is a randomized data structure that maintains a subgraph $G_{\mathcal{M}} \subseteq G$ of a capacitated graph $(G, \boldsymbol{c}_G)$ such that after each of the following operations it provides a ϕ-expander hierarchy $\mathcal{H}_{\mathcal{M}}$ of $(G_{\mathcal{M}}, \boldsymbol{c}_G)$ with height $\eta(\mathcal{H}) \leq k$.

- $\textsc{Init}(G, \boldsymbol{c}_G)$: Given an n-vertex simple capacitated graph $(G, \boldsymbol{c}_G)$, the data structure in expected $T(n)$ time computes a separator $X \subseteq E$ of expected capacity $\mathbb{E}[\boldsymbol{c}_G(X)] \leq \alpha m$ and initializes $G_{\mathcal{M}} \leftarrow G \setminus X$. The output of the subroutine is X.
- $\textsc{Cut}(D)$: Let $\{U_1, \ldots, U_k\}$ be the SCCs of $G_{\mathcal{M}}$. The input is a separator D of $G_{\mathcal{M}}$ such that for each U_i either $D \cap G_{\mathcal{M}}[U_i] = \emptyset$ or $D \cap G_{\mathcal{M}}[U_i] = E_{G_{\mathcal{M}}[U_i]}(S_i, \overline{S_i})$ for some $S_i \subseteq U_i$. The adversary removes D from $G_{\mathcal{M}}$, i.e., it sets $G_{\mathcal{M}} \leftarrow G_{\mathcal{M}} \setminus D$. Let U_D be the union of SCCs that intersect with D. In response, the data structure in expected $T(|U_D|)$ time computes a separator $A \subseteq G_{\mathcal{M}}[U_D]$ of the new $G_{\mathcal{M}}$ of expected capacity $\mathbb{E}[\boldsymbol{c}_G(A)] \leq \beta \boldsymbol{c}_G(D)$ and update $G_{\mathcal{M}} \leftarrow G_{\mathcal{M}} \setminus A$. The output of the subroutine is A.

Our main technical result in this section is the following lemma which says that we can design a $(k+1, \alpha n^{o(1)}\phi, \cdot, \cdot, \cdot)$-hierarchy maintainer using a $(k, \alpha, \cdot, \cdot, \cdot)$-hierarchy maintainer, thus reducing the number of separator edges by a factor $n^{o(1)}\phi \ll \phi^{\Omega(1)}$ for ϕ sufficiently small. The blow-up in the other parameters are carefully set to be manageable.

Lemma A.7.4. *Given a* $(k, \alpha_{\text{prev}}, \beta_{\text{prev}}, \phi_{\text{prev}}, T_{\text{prev}})$*-hierarchy maintainer* $\mathcal{M}_{\text{prev}}$ *for an* n*-vertex simple capacitated graph* $(G, \boldsymbol{c})$*, for any* $L \in \mathbb{N}$ *there exists some* $\delta_L \leq (\log n)^{L^{O(L)}}$ *such that for any* $\phi < O\left(\frac{1}{\delta_L L \beta_{\text{prev}} n^{O(1/L)}}\right)$ *sufficiently small we can construct a* $(k+1, \alpha, \beta, \phi', T)$*-hierarchy maintainer* $\mathcal{M}$ *with*

$$\begin{aligned} \alpha &\leq \phi \cdot \delta_L n^{O(1/L)} \cdot \alpha_{\text{prev}}, \\ \beta &\leq \delta_L n^{O(1/L)}, \\ \phi' &\geq \min\left\{\phi_{\text{prev}}, \frac{\phi}{\delta_L}\right\}, \\ T(n) &= \delta_L n^{O(1/L)} \cdot \widetilde{O}\left(\frac{n^2}{\phi \phi_{\text{prev}}^2} + T_{\text{prev}}(n)\right). \end{aligned} \tag{A.11}$$

Note that the value of β_{prev} only affects the value of ϕ we can choose but not the new β for $\mathcal{M}$. We first show that Lemma A.7.4 in fact already implies Theorem A.7.2, restated below.

Theorem A.7.2. *Given an n-vertex simple capacitated graph $(G, \boldsymbol{c})$ and a parameter $0 < \phi < 2^{-\omega\left(\frac{\log n}{\sqrt{\log\log n}}\right)}$ sufficiently small, there is a randomized $\frac{n^{2+o(1)}}{\phi^3}$ time algorithm that with high probability constructs a $\phi/n^{o(1)}$-expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of $(G, \boldsymbol{c})$ with $\eta = O(\log n)$.*

Proof of Theorem A.7.2. We choose $L = \Theta(\sqrt{\log\log n})$ for which

$$\delta_L \leq (\log n)^{L^{O(L)}} \leq 2^{\log\log n \cdot \Theta(\sqrt{\log\log n})^{\Theta(\sqrt{\log\log n})}} \leq 2^{(\log\log n)^{\Theta(\sqrt{\log\log n})}} = n^{o(1)}.$$

Observe that there is a trivial $(0, 1, 0, 1, O(n^2))$-hierarchy maintainer $\mathcal{M}_0$ which on INIT() simply returns every edge. Since $\phi < o\left(2^{-\frac{\log n}{\sqrt{\log\log n}}}\right)$ is sufficiently small, we have $\phi \leq \left(\frac{1}{\delta_L n^{2/L}}\right)^2$ and $\phi < O\left(\frac{1}{\delta_L L \beta n^{O(1/L)}}\right)$ for $\beta = \delta n^{O(1/L)} \leq n^{o(1)}$. Therefore, starting from $\mathcal{M}_0$, for each $k > 0$ we can apply Lemma A.7.4 on $\mathcal{M}_{k-1}$ to get a $\left(k, \sqrt{\phi}^k, n^{o(1)}, \phi/n^{o(1)}, T_k\right)$-hierarchy maintainer $\mathcal{M}_k$, where $T_k(n) \coloneqq \delta_L^{O(k)} \cdot n^{O(k/L)} \cdot \widetilde{O}(n^2/\phi^3)$. As such, for $\eta = 2\log_{1/\phi}(4n^4) < L$, by calling $\mathcal{M}_\eta$.INIT(G) we get a $\phi/n^{o(1)}$-expander hierarchy of $G \setminus X$ of height $\eta \leq O(\log n)$ for some edge set X with expected size $\mathbb{E}[\boldsymbol{c}_G(X)] \leq \boldsymbol{c}_G(E) \cdot (\sqrt{\phi})^\eta \leq 1/4$. Thus, by Markov's inequality, with probability at least $3/4$ the set X is empty, meaning that the hierarchy we got is indeed a $\phi/n^{o(1)}$-expander hierarchy of G. The expected running time of the algorithm is

$$\log n^{L^{O}(L)} \cdot n^{O(\log_{1/\phi} n)/L} \cdot \widetilde{O}(n^2/\phi^3) = \widehat{O}(n^2/\phi^3)$$

time since $O(\log_{1/\phi} n)/L \leq o(\sqrt{\log\log n})/L = o(1)$. Repeating this $O(\log n)$ time we succeed in worst-case time with high probability. This proves the theorem. □

We now briefly sketch on how we prove Lemma A.7.4. To boost the quality of the maintainer $\mathcal{M}_{\text{prev}}$, let F be the result of running $\mathcal{M}_{\text{prev}}$.INIT$(G)$. Our algorithm essentially takes the k-level hierarchy maintained by $\mathcal{M}_{\text{prev}}$ and constructs the $(k+1)$-th level of it in order to reduce the number of non-expanding edges by roughly a factor of ϕ. Thus, we start with the terminal set F and run expander decomposition in G with respect to F. In other words, the goal is to compute a separator X such that F is ϕ-expanding in $G \setminus X$. If we then have an expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_k)$ of $G \setminus (F \cup X)$ then we can set $X_{k+1} = F$ and obtain an expander hierarchy of $G \setminus X$. Needless to say, we will use $\mathcal{M}_{\text{prev}}$ to maintain such $\mathcal{H}$, and thus throughout the algorithm we need to ensure $\mathcal{H}$ is a hierarchy of $G \setminus (X \cup F)$ by properly calling $\mathcal{M}_{\text{prev}}$.CUT().

To compute an expander decomposition with respect to F, we start with an empty X and let $\mathcal{U}$ be the SCCs of $G_{\mathcal{M}} = G \setminus X$ which is initially $\mathcal{U} = \{V\}$.[26] For each $U \in \mathcal{U}$, we attempt to locate a sparse cut in $G[U]$ via the sparse-cut algorithm we developed in Section A.6. Note that as X is a separator of G and U is strongly connected, $G[U]$ is the same as $G_{\mathcal{M}}[U]$. If we find a sparse cut D, then we include D into X which effectively splits U into (at least) two SCCs on which we recurse our construction. In addition, to ensure that $\mathcal{M}_{\text{prev}}$ holds a hierarchy of $G \setminus (F \cup X)$, we need to call $\mathcal{M}_{\text{prev}}.\textsc{Cut}(D)$ to remove D from $G_{\mathcal{M}_{\text{prev}}}$. Observe that since D is a cut in $G[U]$ and the SCCs of $G_{\mathcal{M}_{\text{prev}}}$ form a refinement of $\mathcal{U}$, the input requirement of $G_{\mathcal{M}_{\text{prev}}}.\textsc{Cut}(D)$ is satisfied. After $\mathcal{M}_{\text{prev}}$ further refines its SCCs and outputs an $A \subseteq G_{\mathcal{M}_{\text{prev}}}[U]$, meaning that now it maintains a hierarchy of $G \setminus (F \cup A \cup X)$, we add A into F to preserve our invariant. Note that doing all these also ensures that we have an expander hierarchy of $G[U] \setminus F$ at all times, which is required by our sparse-cut algorithm. Indeed, if we take the SCCs of $G_{\mathcal{M}_{\text{prev}}}$ contained in U and restrict $\mathcal{H}$ to these SCCs, then we get an expander hierarchy of $G[U]$.

Fact A.7.5. *For a ϕ-expander hierarchy $\mathcal{H} = (D, X_1, \ldots, X_\eta)$ of $(G, \boldsymbol{c}_G)$ and $U \subseteq V$ such that for each $W \in \text{SCC}(G)$ either $W \subseteq U$ or $W \cap U = \emptyset$, the sequence $\mathcal{H}[U] \coloneqq (D \cap G[U], X_1 \cap G[U], \ldots, X_\eta \cap G[U])$ is a ϕ-expander hierarchy of $(G[U], \boldsymbol{c}_G)$.*

Having the overall picture, it now remains to implement the steps efficiently and achieve the desired expected guarantee. For this we adapt and generalize the framework of [HKPW23] which maintains expander decomposition by repeatedly finding sparse cuts and thus fits our purposes well.[27] We give an overview of their framework below.

Overview of [HKPW23]

To certify expansion and locate sparse cuts, [HKPW23] employed a celebrated approach of embedding a witness into each $G[U]$. Informally speaking, a witness is an $\widetilde{\Omega}(1)$-expander with approximately the same degree profile as that of F and is embeddable into $G[U]$ with congestion $\widetilde{O}(1/\phi)$. It may be hard to construct such a witness directly in few calls to the sparse-cut algorithm since we may repeatedly find rather unbalanced sparse cuts which makes the number of iterations $\Omega(n)$. To overcome this, instead of a single witness for $G[U]$, [HKPW23] used a series of witnesses that contain additional *fake* edges that are counted toward the expansion guarantee of the witness yet are not embeddable into $G[U]$.

[26] Here we assume the graph is initially strongly connected.

[27] It is worth mentioning that there is a recent work of [SP24] which improves the almost-linear running time of [HKPW23] to near-linear one. They sidestepped the multi-level approach of [NSW17; BPS20; HKPW23] by a novel *push-pull-relabel* flow algorithm that allowed them to implement the trimming strategy of [SW19]. We leave adapting their framework or our use case as an interesting open direction.

Witness with Fake Edges. While a witness with fake edges does not immediately certify the expansion of the graph, it provides fairly useful information that the graph does not contain a *balanced* sparse cut. Let us start with a threshold R and attempt to construct a witness with R fake edges. In each attempt, we either find a balanced sparse cut of size $\widetilde{\Omega}(R)$ which we then recurse on both sides with a decent size reduction or we certify that no such balanced sparse cut exists in the graph. In the next iteration, we decrease the parameter R to be R' and then repeat the above loop until we certify there is no $\widetilde{\Omega}(R')$-balanced sparse cut either. Crucially, even though as R' decreases we might not get a large size reduction when recursing on both sides of the cut anymore, by setting up the expansion parameter properly in each level, we can in fact guarantee that if we found much more than $\widetilde{\Omega}(R/R')$ sparse cuts in the graph, then there would have been a $\widetilde{\Omega}(R)$-balanced sparse cut in the graph that we start with which contradicts with our R-witness constructed. By setting the R value of a level-ℓ witness to be $n^{\ell/L}$ for some $L = \omega(1)$, the algorithm only needs to run the sparse-cut algorithm $n^{1/L} = n^{o(1)}$ time when computing and maintaining expander decomposition.

Note that there are two notions of levels in our algorithm: One is the level of the expander hierarchy, and we maintain each level of hierarchy with L levels of witnesses. To avoid confusion, in the remainder of this overview the term level means the level of witnesses except when we deliberately use the term *hierarchy level*.

Reparing Witnesses. This idea of using fake edges with decreasing expansion and balance parameters was initiated in [NS17; Wul17] and has been later used either explicitly or implicitly in many other expander decomposition/pruning algorithms [NSW17; BPS20]. What [HKPW23] differs from previous work is the explicit usage of witnesses and the way they set up and repair them which allows for maintaining expander decomposition under updates by directly finding sparse cuts. To be more specific, consider a strongly connected component U for which we want to maintain its expansion. Their algorithm maintains L witnesses $W_0, \ldots, W_L$ with $R_0, \ldots, R_L$ fake edges, where R_ℓ is supposed to be roughly $|U|^{\ell/L}$. For each update to the graph, the algorithm first checks for each witness whether there are edges in it that are embedded into an updated part. If so, these real edges are replaced by fake edges. This increases the number of fake edges in the witnesses which might break the invariant of $R_\ell \approx |U|^{\ell/L}$ (and in particular, if $R_0 \gg 0$, then we have failed to certify the expansion of $G[U]$). Their algorithm thus attempts to repair such an invalid witness from the higher-level witness $W_{\ell+1}$. This is done by setting up a flow problem which corresponds to embedding a sufficiently large number of fake edges in $W_{\ell+1}$ into $G[U]$ so that it becomes valid for level ℓ, and if it fails to do so, the algorithm finds a sparse cut. A careful charging argument is then used in [HKPW23] to bound the total update time throughout a sequence of updates.

Challenges in Adaptation. To adapt their framework to our use case, we replace the standard push-relabel/blocking flow algorithm used in [HKPW23] with the weighted push-relabel algorithm we developed. An immediate challenge for this is that our flow algorithm is not *local* in the sense that it cannot be used to only explore a small neighborhood around the sources. This is in contrast to the classic unweighted push-relabel algorithm which can explore a region with k edges in $O(k)$ time. While the idea of having witnesses with decreasing number of fake edges in principle, as we have touched upon above, allows one to at least intuitively argue that the number of times we need to call sparse-cut algorithm is small, [HKPW23] used a more direct potential-based analysis which heavily relies on their local running time. Thus, we need to apply a different analysis strategy than theirs and prove additional stability properties of their witnesses (see Section A.7.3) which then allow us to incorporate the conceptual guarantee of the high-level fake-edges framework into the specific ways [HKPW23] maintained their witnesses.

Another modification we made to [HKPW23] is the rebuilding strategy. Previously, [HKPW23] attempts to repair a witness whenever it contains too many fake edges. However, this only gives an amortized output recourse which is insufficient for our analysis when interacting with previous hierarchy levels. To overcome this we use a fairly standard strategy: Instead of fixing every witness whenever possible, we set a larger grace period for them and consider a witness valid as long as the number of fake edges it contains falls into its corresponding grace period. We then for each update sample a *random* witness level to be rebuilt. By appropriately setting the sampling probability, we can ensure that (1) each witness will be rebuilt with high probability before it becomes invalid (this is due to the larger grace period we set) and (2) we achieve a worst-case output recourse *in expectation*. While this random rebuilding approach is commonly used, due to the interaction with previous hierarchy levels, during a repair of a witness we might need to abort the current repair, re-sample a higher witness level, and start from there instead. This complication makes the analysis of our expected guarantee fairly cumbersome (see Section A.7.5). We remark that the stability properties we mentioned earlier also play a role in achieving the expected worst-case guarantee.

Organization. In the remainder of the section we present our modification to [HKPW23] with new constructs and analyses which ultimately lead to a proof of Lemma A.7.4. In particular, in Section A.7.2 we apply our flow algorithm of Theorem A.6.1 to construct and repair witnesses. In Section A.7.3 we establish certain stability properties of the witnesses that are key to our analyses. In Section A.7.4 we present the main algorithm of maintaining expander decomposition while interacting with the previous layers via $\mathcal{M}_{\text{prev}}$. In Section A.7.5 we prove that the described algorithm has the desired expected guarantee. Finally, in Section A.7.6 we put everything together and arrive at a proof of Lemma A.7.4 (and hence Theorem A.7.1).

A.7.2 Constructing and Repairing Witnesses

Let $(G, \boldsymbol{c}_G)$ be the input capacitated graph for which we want to construct an expander hierarchy. Throughout the section, we let $m \leq n^4$ denote the sum of capacities of edges in G. We start by giving a formal definition of a witness adapted from [HKPW23, Definition 2.1] and generalized straightforwardly to the capacitated setting.

Definition A.7.6 (R-Witness)**.** For a capacitated simple graph $(W, \boldsymbol{c})$ with $V(W) = V(G)$, a vector $\boldsymbol{r} \in \mathbb{N}^V$, and an embedding $\Pi_{W \to G}$ from W to G, the tuple $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G})$ is an (R, ϕ, ψ)-*out-witness* of $(G, \boldsymbol{c}_G, F)$ for $F \subseteq V \times V$ *with respect to* $\boldsymbol{\gamma} \in \mathbb{N}^V$ if

(1) $\|\boldsymbol{r}\|_1 \leq R$,

(2) $\deg_{F, \boldsymbol{c}_G}(v) \leq \deg_{W, \boldsymbol{c}}(v) + \boldsymbol{r}(v) \leq \frac{1}{\psi} \deg_{F, \boldsymbol{c}_G}(v)$ holds for all $v \in V$,

(3) for every cut $(S, \overline{S})$ with $\boldsymbol{\gamma}(S) \leq \boldsymbol{\gamma}(\overline{S})$ we have $\boldsymbol{c}(E_W(S, \overline{S})) + \boldsymbol{r}(S) \geq \psi(\mathrm{vol}_{W, \boldsymbol{c}_W}(S) + \boldsymbol{r}(S))$, and

(4) $\Pi_{W \to G}$ embeds $(W, \boldsymbol{c})$ into $(G, \boldsymbol{c}_G)$ with congestion $\frac{1}{\phi\psi}$.

If $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G})$ is an (R, ϕ, ψ)-out-witness of $(G, \boldsymbol{c}_G, F)$ and $(\overleftarrow{W}, \boldsymbol{c}, \boldsymbol{r}, \Pi_{\overleftarrow{W} \to G})$ is an (R, ϕ, ψ)-out-witless of $(\overleftarrow{G}, \boldsymbol{c}_G, \overleftarrow{F})$, both with respect to $\boldsymbol{\gamma}$, then $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G})$ is an (R, ϕ, ψ)-*witness* of $(G, \boldsymbol{c}_G, F)$ with respect to $\boldsymbol{\gamma}$.

Note that the $\boldsymbol{r}$ vector corresponds to the concept of fake edges introduced in Section A.7.1.[28] It is easy to see that if $R = 0$, then F is $\Omega(\phi\psi^2)$-expanding in G, and we prove a more general version of this below which says this is even the case for R sufficiently small Claim A.7.9. Oftentimes $\boldsymbol{c}$, $\boldsymbol{r}$, and $\Pi_{W \to G}$ will be clear from context, in which case we may simply refer to W as the witness of $(G, \boldsymbol{c}_G, F)$. In the remainder of the paper we will assume both R and ψ are reasonably bounded by some polynomials in n. More specifically, we assume $R \leq n^{10}$ and $\psi \geq 1/n$. Indeed, the choice of ψ will be made explicit in (A.19), and R is going to be upper-bounded by the total volume of the graph which by capacity scaling is at most n^{10}.

The $\boldsymbol{r}$ vector can be seen as some "fake" edges of the witness which do not exist in the real graph G. While having a witness with many fake edges does not certify that F is expanding in G, it does still shows that there is no *balanced* sparse cut in G with respect to F. A cut S in $(G, \boldsymbol{c}_G)$ is B-*balanced* with respect to F if $\min\{\mathrm{vol}_{F, \boldsymbol{c}_G}(S), \mathrm{vol}_{F, \boldsymbol{c}_G}(\overline{S})\} \geq B$.

[28] That is, each vertex v has $\boldsymbol{r}(v)$ incident fake edges. The $\boldsymbol{r}$ vector is easier to maintain when the graph is updated while suffices for the purpose of [HKPW23]. Indeed, when some vertices S are removed from the graph, it is unclear where in the remaining graph one should add fake edges corresponding to those incident to S; instead, with the $\boldsymbol{r}$ vector one can simply increase $\boldsymbol{r}(v)$ for each remove edge incident to $v \in V \setminus S$.

Claim A.7.7. *If there is a (R, ϕ, ψ)-witness $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W\to G})$ of $(G, \boldsymbol{c}_G, F)$ with respect to any $\boldsymbol{\gamma}$, then there is no $\frac{2R}{\psi}$-balanced $\frac{\phi\psi^2}{2}$-sparse cut in $(G, \boldsymbol{c})$ with respect to F.*

Proof. Consider any cut S with $\frac{2R}{\psi} \le \mathrm{vol}_{F,\boldsymbol{c}_G}(S) \le \mathrm{vol}_{F,\boldsymbol{c}_G}(V \setminus S)$. If $\boldsymbol{\gamma}(S) \le \boldsymbol{\gamma}(V \setminus S)$, then we have

$$\min\{\boldsymbol{c}(E_W(S, V \setminus S)), \boldsymbol{c}(E_W(V \setminus S))\} + \boldsymbol{r}(S) \ge \psi(\mathrm{vol}_{W,\boldsymbol{c}}(S) + \boldsymbol{r}(S)) \ge \psi\mathrm{vol}_{F,\boldsymbol{c}_G}(S) \ge 2R$$

by Definition A.7.6(2) and (3). This implies $\min\{\boldsymbol{c}(E_W(S, V \setminus S)), \boldsymbol{c}(E_W(V \setminus S, S))\} \ge \frac{\psi}{2}\mathrm{vol}_{F,\boldsymbol{c}_G}(S)$, and by the fact that $(W, \boldsymbol{c})$ embeds into $(G, \boldsymbol{c}_G)$ via $\Pi_{W\to G}$ with congestion $\frac{1}{\psi\phi}$ by Definition A.7.6(4), we have $\min\{\boldsymbol{c}_G(E_G(S, V\setminus S)), \boldsymbol{c}_G(E_G(V\setminus S, S))\} \ge \frac{\phi\psi^2}{2}\mathrm{vol}_{F,\boldsymbol{c}_G}(S)$. A symmetric argument applied to the case where $\boldsymbol{\gamma}(S) > \boldsymbol{\gamma}(V\setminus S)$ shows that $\min\{\boldsymbol{c}_G(E_G(S, V\setminus S)), \boldsymbol{c}_G(E_G(V\setminus S, S))\} \ge \frac{\phi\psi^2}{2}\mathrm{vol}_{F,\boldsymbol{c}_G}(V\setminus S) \ge \frac{\phi\psi^2}{2}\mathrm{vol}_{F,\boldsymbol{c}_G}(S)$ as well. Therefore, such an S can not be a $\frac{\phi\psi^2}{2}$-sparse cut. □

Claim A.7.8. *If a (not necessarily strongly connected) subgraph $G[U]$ has volume $\mathrm{vol}_{F,\boldsymbol{c}_G}(G[U]) < 1/\phi$, then F is ϕ-expanding in $(G[U], \boldsymbol{c}_G)$.*

Proof. It suffices to consider the case when $G[U]$ is strongly connected by the definition of ϕ-expanding. Since $\boldsymbol{c}_G(E_{G[U]}(S, U \setminus S)) \ge 1$ for all $S \subseteq U$ and $\mathrm{vol}_{F,\boldsymbol{c}_G}(S), \mathrm{vol}_{F,\boldsymbol{c}_G}(\overline{S}) < 1/\phi$, we have F is ϕ-expanding in $(G[U], \boldsymbol{c}_G)$. □

Claim A.7.9. *If there is a (R, ϕ, ψ)-witness for $(G, \boldsymbol{c}_G, F)$ with respect to any $\boldsymbol{\gamma}$ where $R < 1/\phi$, then F is $\frac{\phi\psi^2}{2}$-expanding in $(G, \boldsymbol{c}_G)$.*

Proof. The existence of a (R, ϕ, ψ)-witness by Claim A.7.7 implies that there is no $\frac{2R}{\psi}$-balanced $\frac{\phi\psi^2}{2}$-sparse cut in $(G, \boldsymbol{c}_G)$ with respect to F. This suggests that G can only have one strongly connected component with volumes at least $\frac{2R}{\psi} \le \frac{2}{\phi\psi}$, otherwise there is a cut with no edges that separate two such components which would lead to a contradiction. By Claim A.7.8, F is $\frac{\phi\psi}{2}$-expanding in those components with small volume. On the other hand, consider the only component U in G that has volume at least $\frac{2}{\phi\psi}$. Note that Claim A.7.7 also suggests that there is no $\frac{2R}{\psi}$-balanced $\frac{\phi\psi^2}{2}$-sparse cut in $(G[U], \boldsymbol{c}_G)$ with respect to F. Indeed, the same cut would have been $\frac{2R}{\psi}$-balanced $\frac{\phi\psi^2}{2}$-sparse in $(G, \boldsymbol{c}_G)$ as well if it existed. However, if a cut in $(G[U], \boldsymbol{c}_G)$ has volume less than $\frac{2}{\phi\psi}$, then it can never be $\frac{\phi\psi^2}{2}$-sparse since $G[U]$ is strongly connected. This shows that F is $\frac{\phi\psi^2}{2}$-expanding in $(G[U], \boldsymbol{c}_G)$ as well, hence in $(G, \boldsymbol{c}_G)$. □

Algorithms for Constructing R-Witnesses. We will have two primitives which tries to construct R-witnesses for some set of terminal edges F, both which are based on our sparse cut algorithm from Section A.6.

- CutOrEmbed which either finds a $\widetilde{\Omega}(R)$-balanced sparse cut or an R-witness. This is done by running a standard cut-matching game.
- PruneOrRepair which takes an R-witness as input, and either finds a $\widetilde{\Omega}(R')$-balanced sparse cut or an R'-witness for some $R' \leq R$. This is done by attempting to embed the fake edges in the R-witness further into the graph.

Let $c_{A.6.1} \in \mathbb{N}$ be a universal constant such that the cut Theorem A.6.1 returns satisfies

$$\boldsymbol{c}(E_G(S,\overline{S})) \leq \frac{c_{A.6.1} \cdot |\boldsymbol{f}| + \mathrm{vol}_{F,\boldsymbol{c}}(S)}{\kappa}.$$

Let $z := 20 \log n$ be fixed throughout the rest of the section.

Lemma A.7.10 (Analogous to [HKPW23, Lemma 3.1])**.** *Given an n-vertex strongly connected simple capacitated graph $(G, \boldsymbol{c}_G)$, terminal edge set $F \subseteq E$, vectors $\boldsymbol{r}, \boldsymbol{\gamma} \in \mathbb{Z}^V_{\geq 0}$, an (R, ϕ, ψ)-witness $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G})$ of $(G, \boldsymbol{c}_G, F)$ with respect to $\boldsymbol{\gamma}$, a parameter $R' \geq 0$ such that $R' \leq R \leq \frac{\psi}{8z}\mathrm{vol}_{F,\boldsymbol{c}_G}(V)$, and a ϕ'-expander hierarchy $\mathcal{H}$ of $(G \setminus F, \boldsymbol{c}_G)$ with height $O(\log n)$, there is an algorithm PruneOrRepair$(G, F, W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G}, \phi, \psi, R', \mathcal{H})$ that either outputs*

1. *a set $S \subseteq V$ with $\frac{\psi}{16z} \cdot R' \leq \mathrm{vol}_{F,\boldsymbol{c}_G}(S) + \boldsymbol{r}(S) \leq \frac{8z}{\psi} \cdot R$ such that $\boldsymbol{c}_G(E_G(S,\overline{S})) < \frac{\phi\psi^3}{256} \cdot (\mathrm{vol}_{F,\boldsymbol{c}_G}(S) + \boldsymbol{r}(S))$ or*
2. *an (R', ϕ, ψ')-out-witness $(W', \boldsymbol{c}', \boldsymbol{r}', \Pi_{W' \to G})$ of $(G, \boldsymbol{c}_G, F)$ with respect to $\boldsymbol{\gamma}$, where $\psi' := \frac{\psi^4}{2048 c_{A.6.1} z^2} = \Omega\left(\frac{\psi^4}{\log^2 n}\right)$.*

The algorithm runs in time $\widetilde{O}\left(\frac{n^2}{\phi\phi'^2\psi^4}\right)$.[29]

Proof. We follow essentially the same proof strategy as [HKPW23, Lemma 3.1] but with parameters tailored to our needs. We set up a diffusion instance $\mathcal{I} = (G, \boldsymbol{c}_G, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ with $\boldsymbol{\Delta} := z \cdot \frac{8}{\psi} \cdot \boldsymbol{r}$ and $\boldsymbol{\nabla} := \deg_{F,\boldsymbol{c}_G} + \boldsymbol{r}$. Note that $\mathcal{I}$ is a diffusion instance as $R \leq \frac{\psi}{8z}\mathrm{vol}_{F,\boldsymbol{c}}(V)$. Also note that $z \geq \log \|\boldsymbol{\Delta}\|_1$. We are going to compute a flow $\boldsymbol{f}^*$ routing $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in G with congestion κz for $\kappa := \frac{1024 \cdot c_{A.6.1} \cdot z}{\phi\psi^4}$ by invoking Theorem A.6.1 z times. Let $\boldsymbol{f}^* := \mathbf{0}$. While $\|\boldsymbol{\Delta}\|_1 > R'$, we run SparseCut$(\mathcal{I}, \kappa, F, \mathcal{H})$ in Theorem A.6.1 to find a flow $\boldsymbol{f}$. If $\boldsymbol{f}$ routes half of the demand, i.e., $|\boldsymbol{f}| \geq \frac{1}{2}\|\boldsymbol{\Delta}\|_1$, then we update $\boldsymbol{f}^* \leftarrow \boldsymbol{f}^* + \boldsymbol{f}$ and set $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}_{\boldsymbol{f}}$ to be the residual supply while keeping the sink intact and then repeat. Otherwise, we have $\mathbf{ex}_{\boldsymbol{f}}(V) > \frac{1}{2}\|\boldsymbol{\Delta}\|_1 > \frac{1}{2}R'$ and $|\boldsymbol{f}| < \frac{1}{2}\|\boldsymbol{\Delta}\|_1$. In this case we will find a sparse cut and output it in Case 1. Let S be the cut outputted by Theorem A.6.1

[29] Note that as in [HKPW23], the algorithm does not need to take $\boldsymbol{\gamma}$ as an input.

on this invocation. We have $\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}(S)\geq \boldsymbol{r}(S)\geq \frac{\psi}{8z}\mathbf{ex}_{\boldsymbol{f}}(S)\geq \frac{\psi}{16z}R'$ and $\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}(S)=\boldsymbol{\nabla}(S)=\mathbf{abs}_{\boldsymbol{f}}(S)\leq \frac{8z}{\psi}R$. Also, Theorem A.6.1 asserts that

$$\begin{aligned}\boldsymbol{c}_G(E_G(S,\overline{S}))&\leq \frac{c_{A.6.1}\cdot|\boldsymbol{f}|+\mathrm{vol}_{F,\boldsymbol{c}_G}(S)}{\kappa}<\frac{c_{A.6.1}\cdot\frac{4z}{\psi}\boldsymbol{r}(S)+\mathrm{vol}_{F,\boldsymbol{c}_G}(S)}{\kappa}\\&\leq\frac{\phi\psi^3}{256}\cdot(\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}(S))\end{aligned}$$

where we used that $\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}(S)\geq\frac{\psi}{8z}\boldsymbol{\Delta}(S)$ and $|\boldsymbol{f}|<\frac{1}{2}\|\Delta\|_1\leq\frac{4z}{\psi}\boldsymbol{r}(S)$. This shows that S indeed satisfies the output requirement of Lemma A.7.10.

If none of the calls to Theorem A.6.1 routes less than half of the demand, we end up with a flow $\boldsymbol{f}^*$ with $\mathbf{ex}_{\boldsymbol{f}^*}(V)\leq R'$ and congestion κz. In this case we can construct a new witness $(W',\boldsymbol{c}',\boldsymbol{r}',\Pi_{W'\to G})$ in Case 2 as follows. Initialize W' as W, (and $\Pi_{W'\to G}$ as $\Pi_{W\to G}$ consequently), $\boldsymbol{c}'$ as $\boldsymbol{c}$, and $\boldsymbol{r}'$ as $\frac{8}{\psi}\boldsymbol{r}$. By a standard flow decomposition argument, we can in $\widetilde{O}(n^2)$ time decompose $\boldsymbol{f}^*$ into at most n^2 flow paths P_i that sends c_i units of flow from (u_i,v_i). For each such flow path P_i, we add an edge (u_i,v_i) with capacity $\boldsymbol{c}'(u_i,v_i)=c_i$ to W', merging parallel edges if exists. We then decrease $\boldsymbol{r}'(u_i)$ by c_i. We argue that $(W',\boldsymbol{c}',\boldsymbol{r}',\Pi_{W'\to G})$ forms an (R',ϕ,ψ')-out-witness of $(G,\boldsymbol{c}_G,F)$ with respect to $\boldsymbol{\gamma}$.

Properties (1) and (4). The fact that $\|\boldsymbol{r}'\|_1\leq R'$ is by definition. The new embedding $\Pi_{W'\to G}$ has congestion at most $\frac{1}{\phi\psi}+z\cdot\frac{1024\cdot c_{A.6.1}\cdot z}{\phi\psi^4}\leq\frac{1}{\phi\psi'}$.

Property (2). Observe that $\deg_{W',\boldsymbol{c}'}(v)+\boldsymbol{r}'(v)$ is initialized to $\deg_{W,\boldsymbol{c}}(v)+\frac{8z}{\psi}\cdot\boldsymbol{r}(v)\in\left[\deg_{F,\boldsymbol{c}_G}(v),\ \frac{8z}{\psi^2}\cdot\deg_{F,\boldsymbol{c}_G}(v)\right]$. Also note that each v absorbs at most $z\cdot\boldsymbol{\nabla}(v)=z(\deg_{F,\boldsymbol{c}_G}(v)+\boldsymbol{r}(v))$ units of demand. For each flow path P_i from u_i to v_i with c_i units of flow, $\deg_{W',\boldsymbol{c}'}(u_i)+\boldsymbol{r}'(u_i)$ stays the same while $\deg_{W',\boldsymbol{c}'}(v)+\boldsymbol{r}'(v)$ increases by c_i. As a result, we have

$$\deg_{W',\boldsymbol{c}'}(v)+\boldsymbol{r}'(v)\leq\frac{8z}{\psi^2}\cdot\deg_{F,\boldsymbol{c}_G}(v)+z(\deg_{F,\boldsymbol{c}_G}(v)+\boldsymbol{r}(v)) \tag{A.12}$$

$$\overset{(*)}{\leq}\frac{10z}{\psi^2}\cdot\deg_{F,\boldsymbol{c}_G}(v) \tag{A.13}$$

$$\leq\frac{1}{\psi'}\deg_{F,\boldsymbol{c}_G}(v). \tag{A.14}$$

Note that as in [HKPW23] we will use the stronger bound of $(*)$ later.

Property (3). Consider a cut S where $\boldsymbol{\gamma}(S)\leq\boldsymbol{\gamma}(\overline{S})$ for which $\boldsymbol{c}(E_W(S,\overline{S}))+\boldsymbol{r}(S)\geq\psi(\mathrm{vol}_{W,\boldsymbol{c}}(S)+\boldsymbol{r}(S))$.

- If $\boldsymbol{c}(E_W(S,\overline{S})) \geq \boldsymbol{r}(S)$: We have $\mathrm{vol}_{W',\boldsymbol{c}'}(S) + \boldsymbol{r}'(S) \leq \frac{10z}{\psi^2}\mathrm{vol}_{F,\boldsymbol{c}_G}(S) \leq \frac{10z}{\psi^2}(\mathrm{vol}_{W,\boldsymbol{c}}(S) + \boldsymbol{r}(S))$ by (A.14). This implies

$$\begin{aligned}\boldsymbol{c}'(E_{W'}(S,\overline{S})) + \boldsymbol{r}'(S) &\geq \boldsymbol{c}(E_W(S,\overline{S})) \geq \frac{1}{2}(\boldsymbol{c}(E_W(S,\overline{S})) + \boldsymbol{r}(S)) \\ &\geq \frac{\psi}{2}(\mathrm{vol}_{W,\boldsymbol{c}}(S) + \boldsymbol{r}(S)) \geq \frac{\psi^3}{20z}(\mathrm{vol}_{W',\boldsymbol{c}'}(S) + \boldsymbol{r}'(S)).\end{aligned}$$

- If $\boldsymbol{c}(E_W(S,\overline{S})) < \boldsymbol{r}(S)$ and $\boldsymbol{r}'(S) > \frac{1}{2}\boldsymbol{r}(S)$: We have

$$\begin{aligned}\boldsymbol{c}'(E_{W'}(S,\overline{S})) + \boldsymbol{r}'(S) &\geq \frac{1}{2}(\boldsymbol{c}(E_W(S,\overline{S})) + \boldsymbol{r}(S)) \geq \frac{\psi}{2}(\mathrm{vol}_{W,\boldsymbol{c}}(S) + \boldsymbol{r}(S)) \\ &\geq \frac{\psi^3}{20z}(\mathrm{vol}_{W',\boldsymbol{c}'}(S) + \boldsymbol{r}'(S)).\end{aligned}$$

- If $\boldsymbol{c}(E_W(S,\overline{S})) < \boldsymbol{r}(S)$ and $\boldsymbol{r}'(S) \leq \frac{1}{2}\boldsymbol{r}(S)$: The flow paths with tails in S send precisely $\frac{8z}{\psi}\boldsymbol{r}(S) - \boldsymbol{r}'(S) \geq \frac{4z}{\psi}\boldsymbol{r}(S)$ units of flow, within which at most $z \cdot \boldsymbol{\nabla}(S) = z(\mathrm{vol}_{F,\boldsymbol{c}_G}(S) + \boldsymbol{r}(S))$ units are absorbed in S Since each path P_i with c_i units of flow from S to $\overline{S}$ turn into an edge of capacity c_i in $E_{W'}(S,\overline{S})$, we have $\boldsymbol{c}'(E_{W'}(S,\overline{S})) \geq \frac{4z}{\psi}\boldsymbol{r}(S) - z(\mathrm{vol}_{F,\boldsymbol{c}_G}(S) + \boldsymbol{r}(S))$. We can bound $\mathrm{vol}_{F,\boldsymbol{c}_G}(S) \leq \mathrm{vol}_{W,\boldsymbol{c}}(S) + \boldsymbol{r}(S) \leq \frac{1}{\psi}(\boldsymbol{c}(E_W(S, V \setminus S)) + \boldsymbol{r}(S)) \leq \frac{2}{\psi}\boldsymbol{r}(S)$, which then gives $\boldsymbol{c}'(E_{W'}(S, V \setminus S)) \geq \frac{2z}{\psi}\boldsymbol{r}(S) \geq z \cdot \mathrm{vol}_{F,\boldsymbol{c}_G}(S) \geq \frac{\psi^2}{10z^2}(\mathrm{vol}_{W',\boldsymbol{c}'}(S) + \boldsymbol{r}'(S))$.

As $\psi' \leq \min\left\{\frac{\psi^3}{20z}, \frac{\psi^2}{10z^2}\right\}$, Property (3) is preserved. Since the running time of the algorithm is dominated by $O(\log n)$ calls to Theorem A.6.1 which runs in $\widetilde{O}\left(\frac{n^2}{\phi\phi'^2\psi^4}\right)$, the proof is completed. □

By running the cut-matching game of Theorem A.3.6 with our sparse-cut algorithm of Theorem A.6.1, we can also obtain the following lemma which constructs an initial witness. As the proof is fairly standard, we defer it to Section A.10.

Lemma A.7.11. *Given an n-vertex strongly connected graph $G = (V, E)$, terminal edge set $F \subseteq E$, parameters ϕ, R, and a ϕ'-expander hierarchy $\mathcal{H}$ of $G \setminus F$ with height $O(\log n)$, there is an algorithm* CutOrEmbed$(G, \boldsymbol{c}_G, F, \phi, R')$ *that either output*

1. *a set $S \subseteq V$ such that $\min\{\boldsymbol{c}_G(E_G(S,\overline{S})), \boldsymbol{c}_G(E_G(\overline{S},S))\} < \phi \cdot \mathrm{vol}_{F,\boldsymbol{c}_G}(S)$ and $\frac{1}{4t_{CMG}}R \leq \mathrm{vol}_{F,\boldsymbol{c}_G}(S) \leq \frac{1}{2}\mathrm{vol}_{F,\boldsymbol{c}_G}(V)$ or*
2. *a $\boldsymbol{\gamma} \in \mathbb{N}^V$ and an $(R, \phi, \widetilde{\psi})$-witness $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W\to G})$ of $(G, \boldsymbol{c}_G, F)$ where $\widetilde{\psi} = \Omega\left(\frac{1}{\log^3 n}\right)$ with respect to $\boldsymbol{\gamma}$.*

The algorithm runs in time $\widetilde{O}\left(\frac{n^2}{\phi\phi'^2}\right)$

A.7.3 Stability of Witnesses

Following the framework of [HKPW23], our algorithm maintains for each strongly connected component U of the current graph and each $\ell \in \{0, \dots, L\}$ a witness $(W_{U,\ell}, \boldsymbol{c}_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell} \to G[U]})$. To maintain these witnesses, we will repeatedly find sparse cuts in the graph and remove them until we have certified such cuts do not exist. After each cut is found, we may also update the volume with which the witness needs to certify in response to $\mathcal{M}_{\text{prev}}.\textsc{Cut}()$. We handle these updates through the subroutine $\textsc{UpdateWitness}(U, S, A)$ implemented in Algorithm A.3. The subroutine removes the cut S from U and then increases the terminal set from F to $F \cup A$, while making sure that all the $W_{U,\ell}$'s remain valid witnesses.

Each cut S that we call $\textsc{UpdateWitness}()$ on will either correspond directly to a call to the $\textsc{Cut}()$ function in Definition A.7.3, in which case we refer to S as an *external* cut, or correspond to the sparse cut found internally when maintaining witnesses (specifically through Lemmas A.7.10 and A.7.11), in which case we refer to S as an *internal* cut. To handle the updates, the algorithm simply projects each witness $W_{U,\ell}$ onto $U \setminus S$ and removes edges in it that are no longer embedded into the new $G[U \setminus S]$. It also makes necessary increases to the value of $\boldsymbol{r}_{U,\ell}(v)$ when new edges are added to F to ensure the validity of the witnesses. Note that we essentially ignore S and are not projecting $W_{U,\ell}$ onto it. This is because once $\textsc{UpdateWitness}(U, S, A)$ is called, as we shall see in Algorithm A.5, our algorithm will immediately reconstruct all the witnesses of S entirely from scratch.

We take a rather modularized approach and guarantee that all witnesses, once constructed or repaired, will only be updated using $\textsc{UpdateWitness}()$. Therefore, before giving the full details on how the subroutine is used in Section A.7.4 and how the parameters are set, we first establish several stability properties that are key to our analysis later.

First note that each $W_{U,\ell}$ remains a valid witness, albeit with an increase in $\|\boldsymbol{r}_{U,\ell}\|_1$.

Claim A.7.12. *After each call to* $\textsc{UpdateWitness}()$*, the tuple* $(W_{U,\ell}, \boldsymbol{c}_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell} \to G[U]})$ *remains a valid* $(\infty, \phi, \psi_\ell)$*-witness of* $(G[U], \boldsymbol{c}_G, F)$ *with respect to* $\boldsymbol{\gamma}_U$*.*

Proof. Observe that for each edge $e' = (u, v)$ removed from $W_{U,\ell}$, there is a corresponding increase in $\boldsymbol{r}_{U,\ell}(u)$ and $\boldsymbol{r}_{U,\ell}(v)$ by $\boldsymbol{c}_{U,\ell}(e')$. Likewise, each newly added terminal edge (u, v) has a corresponding increase in $\boldsymbol{r}_{U,\ell}(u)$ and $\boldsymbol{r}_{U,\ell}(u)$ by $\boldsymbol{c}_G(u, v)$. Let U' and F' be the set U and F before the update. Let $W'_{U',\ell}$, $\boldsymbol{c}'_{U',\ell}$, and $\boldsymbol{r}'_{U',\ell}$ be the old witness. We thus have for each $u \in U$ that

$$\left(\deg_{W_{U,\ell}, \boldsymbol{c}_{U,\ell}}(v) + \boldsymbol{r}_{U,\ell}(v)\right) - \left(\deg_{W'_{U',\ell}, \boldsymbol{c}_{U,\ell}}(v) + \boldsymbol{r}'_{U',\ell}(v)\right) = (\deg_{F, \boldsymbol{c}_G}(v) - \deg_{F', \boldsymbol{c}_G}(v)). \tag{A.15}$$

Thus, Property (2) is preserved. Property (4) is trivially preserved as well since the congestion of $\Pi_{W_{U,\ell} \to G[U]}$ can only decrease. For Property (3), consider a cut $(T, U \setminus T)$ in the new U with $\boldsymbol{\gamma}_U(T) \le \boldsymbol{\gamma}_U(U \setminus T)$. As $\boldsymbol{\gamma}_{U'}(T) \le \boldsymbol{\gamma}_{U'}((U \setminus T) \cup S)$

Algorithm A.3: Implementation of UpdateWitness()

global : the terminal edge set F

function UpdateWitness($U, S \subseteq U, A \subseteq G[U]$)
  // remove S from U and add A into the terminal set F
  for $\ell \in \{0, \ldots, L\}$ **do**
    for $e \in E_{G[U]}(S, \overline{S}) \cup E_{G[U]}(S, \overline{S})$ *and* $e' \in \Pi^{-1}_{W_{U,\ell} \to G[U]}(e)$ **do**
      Let $e' = (u, v)$. Increase $\boldsymbol{r}_{U,\ell}(u)$ and $\boldsymbol{r}_{U,\ell}(v)$ by $\boldsymbol{c}_{U,\ell}(e')$ and remove e' from $W_{U,\ell}$.
    for $e = (u, v) \in A \setminus F$ **do**
      Increase $\boldsymbol{r}_{U,\ell}(u)$ and $\boldsymbol{r}_{U,\ell}(v)$ by $\boldsymbol{c}_G(e)$.
  Replace U in $\mathcal{U}$ with $U \setminus S$ and S.
  Let $W_{U\setminus S,\ell}$ be $W_{U,\ell}[U \setminus S]$, $\boldsymbol{r}_{U\setminus S,\ell}$ be $\boldsymbol{r}_{U\setminus S,\ell}$ restricted to $U \setminus S$, and $\boldsymbol{\gamma}_{U\setminus S}$ be $\boldsymbol{\gamma}_U$ restricted to $U \setminus S$.
  Update $U \leftarrow U \setminus S$ and $F \leftarrow F \cup A$.

clearly holds, we have

$$\boldsymbol{c}_{U,\ell}\left(E_{W'_{U',\ell}}(T, (U \setminus T) \cup S)\right) + \boldsymbol{r}'_{U',\ell}(T) \geq \psi(\mathrm{vol}_{W'_{U',\ell},\boldsymbol{c}_{U,\ell}}(T) + \boldsymbol{r}'_{U',\ell}(T))$$

by the properties of the old witness. Using the above arguments we can derive

$$\begin{aligned}
&\boldsymbol{r}_{U,\ell}(T) - \boldsymbol{r}'_{U',\ell}(T) \\
&\quad \geq \left(\boldsymbol{c}_{U,\ell}\left(E_{W'_{U',\ell}}(T, (U \setminus T) \cup S)\right) - \boldsymbol{c}_{U,\ell}\left(E_{W_{U,\ell}}(T, U \setminus T)\right)\right) \\
&\qquad + (\mathrm{vol}_{F,\boldsymbol{c}_G}(T) - \mathrm{vol}_{F',\boldsymbol{c}_G}(T))
\end{aligned}$$

which implies

$$\begin{aligned}
&\boldsymbol{c}_{U,\ell}(E_{W_{U,\ell}}(T, U \setminus T)) + \boldsymbol{r}_{U,\ell}(T) \\
&\quad \geq \psi(\mathrm{vol}_{W'_{U',\ell},\boldsymbol{c}_{U,\ell}}(T) + \boldsymbol{r}'_{U',\ell}(T)) + (\mathrm{vol}_{F,\boldsymbol{c}_G}(T) - \mathrm{vol}_{F',\boldsymbol{c}_G}(T)) \\
&\quad \geq \psi(\mathrm{vol}_{W_{U,\ell},\boldsymbol{c}_{U,\ell}}(T) + \boldsymbol{r}_{U,\ell}(T))
\end{aligned}$$

when combined with (A.15). This proves that $(W_{U,\ell}, \boldsymbol{c}_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell} \to G[U]})$ is an (∞, ϕ, ψ)-out-witness of $(G[U], \boldsymbol{c}_G, F)$. The proof on the reversed graph follows analogously. □

We further argue that the increase in $\|\boldsymbol{r}_{U,\ell}\|_1$ is relatively stable with respect to internal cuts and is mostly dominated by external cuts. In particular, we consider the following scenario.

Scenario A.7.13. Suppose at some moment $(W_{U,\ell}, \boldsymbol{c}_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell}\to G[U]})$ is an (R, ϕ, ψ_ℓ)-witness of $(G[U], \boldsymbol{c}_G, F)$. Let U_0 be the set U and F_0 be the set F at this moment. There is then a sequence of r calls to UpdateWitness(U, S_i, A_i) for $S_i \subseteq U_{i-1}$ (where $U_i \coloneqq U_{i-1} \setminus S_i$) and $A_i \subseteq G[U_{i-1}]$ such that the cut S_i has $\delta_i \coloneqq \min\{\boldsymbol{c}_G(E_{G[U_{i-1}]}(S_i, \overline{S_i})), \boldsymbol{c}_G(E_{G[U_{i-1}]}(\overline{S_i}, S_i))\}$ boundary capacities and there are at most $\Delta_i = 2\boldsymbol{c}_G(A_i)$ units of volume added after the i-th update. Let $F_i \coloneqq F_{i-1} \cup A_i$ be the terminal edge set after the i-th update.

Remark A.7.14. Note that Scenario A.7.13 models the case when these are the only changes made to $W_{U,\ell}$. Later in Section A.7.4 we will perform periodic reconstruction of $W_{U,\ell}$ which is not characterized by Scenario A.7.13, and we will ensure that the stability properties established in this section are used only when Scenario A.7.13 applies.

We first bound how much $\|\boldsymbol{r}_{U,\ell}\|_1$ can grow in terms of the δ_i's assuming there is no terminal addition at all, i.e., $A_i = \emptyset$ for all $i \in [r]$. We make the following observation regarding the boundary edges of a union of cuts.

Observation A.7.15. *Suppose there is a sequence of cuts $S_1, \ldots, S_k$ in a graph $G = (V, E)$ with $S_i \subseteq V_{i-1}$ where $V_0 \coloneqq V$ and $V_i \coloneqq V_{i-1} \setminus S_i$ and consider $S \coloneqq \bigcup_{j\in\mathcal{J}} S_j$ for some $\mathcal{J} \subseteq [k]$. Then, we have*

$$E_G(S, \overline{S}) \subseteq \bigcup_{i\in\mathcal{J}} E_{G[V_{i-1}]}(S_i, \overline{S_i}) \cup \bigcup_{i\notin\mathcal{J}} E_{G[V_{i-1}]}(\overline{S_i}, S_i).$$

Lemma A.7.16. *In Scenario A.7.13, suppose $A_i = \emptyset$ for all $i \in [r]$, then after the r updates we have $\|\boldsymbol{r}_{U_r,\ell}\|_1 \le \frac{3R}{\psi_\ell} + \frac{4}{\psi_\ell^2\phi}\sum_{j\in[r]}\delta_j$.*

Proof. Let us call a cut S_i *out-sparse* if $\boldsymbol{c}_G(E_{G[U_{i-1}]}(S_i, \overline{S_i})) \le \boldsymbol{c}_G(E_{G[U_{i-1}]}(\overline{S_i}, S_i))$ and *in-sparse* otherwise. Let

$$S_{\text{out}} \coloneqq \bigcup_{S_j \text{ is out-sparse}} S_j \quad \text{and} \quad S_{\text{in}} \coloneqq \bigcup_{S_j \text{ is in-sparse}} S_j,$$

be cuts in $G[U_0]$, for which by Observation A.7.15 we have

$$\boldsymbol{c}_G\left(E_{G[U_0]}(S_{\text{out}}, \overline{S_{\text{out}}})\right) \le \sum_{j\in[r]}\delta_j \quad \text{and} \quad \boldsymbol{c}_G\left(E_{G[U_0]}(\overline{S_{\text{in}}}, S_{\text{in}})\right) \le \sum_{j\in[r]}\delta_j. \tag{A.16}$$

Let $D \coloneqq E(G[U_0]) \setminus E(G[U_r])$ be the set of edges that are deleted from $G[U]$ after the r updates. Note that the each increase in $\boldsymbol{r}_{U_r,\ell}(v)$ from the initial $\boldsymbol{r}_{U_0,\ell}(v)$ for $v \in U_r$ corresponds to an edge e incident to v in $W_{U_0,\ell}$ that embeds into a deleted edge, i.e., $\Pi_{W_{U_0,\ell}\to G[U_0]}(e) \cap D \neq \emptyset$. Let $D^{-1} \subseteq E(W_{U_0,\ell})$ be the set of such edges. We can analyze the size of $|D^{-1}|$ by considering an $e \in D^{-1}$.

- If both endpoints of e are in U_k, then $\Pi_{W_{U_0,\ell}}(e)$ must use one edge in $E_{G[U_0]}(S_{\text{out}}, \overline{S_{\text{out}}}) \cup E_{G[U_0]}(\overline{S_{\text{in}}}, S_{\text{in}})$ since it must enters and leaves one of S_{out} and S_{in}.
- If $e \in E_{W_{U_0,\ell}}(S_{\text{out}}, \overline{S_{\text{out}}})$, then $\Pi_{W_{U_0,\ell} \to G[U_0]}(e) \cap E_{G[U_0]}(S_{\text{out}}, \overline{S_{\text{out}}}) \neq \emptyset$; similarly, if $e \in E_{W_{U_0,\ell}}(\overline{S_{\text{in}}}, S_{\text{in}})$, then $\Pi_{W_{U_0,\ell} \to G[U_0]}(e) \cap E_{G[U_0]}(\overline{S_{\text{in}}}, S_{\text{in}}) \neq \emptyset$.
- Otherwise, we have $e \in E_{W_{U_0,\ell}}(\overline{S_{\text{out}}}, S_{\text{out}})$ or $e \in E_{W_{U_0,\ell}}(S_{\text{in}}, \overline{S_{\text{in}}})$.

This gives us the bound of

$$\begin{aligned}\boldsymbol{c}_{U,\ell}(D^{-1}) \leq \frac{1}{\psi_\ell \phi} \cdot \left(\boldsymbol{c}_G\left(E_{G[U_0]}(S_{\text{out}}, \overline{S_{\text{out}}})\right) + \boldsymbol{c}_G\left(E_{G[U_0]}(\overline{S_{\text{in}}}, S_{\text{in}})\right)\right) \\ + \boldsymbol{c}_{U,\ell}\left(E_{W_{U_0,\ell}}(\overline{S_{\text{out}}}, S_{\text{out}})\right) + \boldsymbol{c}_{U,\ell}\left(E_{W_{U_0,\ell}}(S_{\text{in}}, \overline{S_{\text{in}}})\right)\end{aligned}$$

by the congestion of $\Pi_{W_{U_0,\ell} \to G[U_0]}$ from Property (4). To bound the right-hand side, we prove the following claim.

Claim A.7.17. *For any (R, ϕ, ψ)-witness $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G})$ we have $\boldsymbol{c}(E_W(\overline{S}, S)) \leq \frac{1}{\psi}(\boldsymbol{c}(E_W(S, \overline{S})) + R)$ for all $S \subseteq V$.*

Proof. Note that $\boldsymbol{c}(E_W(\overline{S}, S)) \leq \min\{\text{vol}_{W,\boldsymbol{c}}(S), \text{vol}_{W,\boldsymbol{c}}(\overline{S})\}$ and thus it suffices to bound the latter. If $\boldsymbol{\gamma}(S) \leq \boldsymbol{\gamma}(\overline{S})$, then since W is an out-witness we have $\text{vol}_{W,\boldsymbol{c}}(S) \leq \text{vol}_{W,\boldsymbol{c}}(S) + \boldsymbol{r}(S) \leq \frac{1}{\psi}(\boldsymbol{c}(E_W(S, \overline{S})) + \boldsymbol{r}(S)) \leq \frac{1}{\psi}(\boldsymbol{c}(E_W(S, \overline{S})) + R)$. Similarly, if $\boldsymbol{\gamma}(S) > \boldsymbol{\gamma}(\overline{S})$, then since $\overleftarrow{W}$ is an out-witness we have $\text{vol}_{W,\boldsymbol{c}}(\overline{S}) \leq \text{vol}_{W,\boldsymbol{c}}(\overline{S}) + \boldsymbol{r}(\overline{S}) \leq \frac{1}{\psi}(\boldsymbol{c}(E_{\overleftarrow{W}}(\overline{S}, S)) + \boldsymbol{r}(\overline{S})) \leq \frac{1}{\psi}(\boldsymbol{c}(E_W(S, \overline{S})) + R)$. □

Following Claim A.7.17, we have

$$\begin{aligned}\boldsymbol{c}(D^{-1}) &\leq \frac{2}{\psi_\ell \phi} \cdot \sum_{j \in [r]} \delta_j + \frac{1}{\psi_\ell}\left(\boldsymbol{c}_{U_0,\ell}\left(E_{W_{U_0,\ell}}(S_{\text{out}}, \overline{S_{\text{out}}})\right) + R\right) \\ &\quad + \frac{1}{\psi_\ell}\left(\boldsymbol{c}_{U_0,\ell}\left(E_{W_{U_0,\ell}}(\overline{S_{\text{in}}}, S_{\text{in}})\right) + R\right) \\ &\leq \frac{4}{\psi_\ell^2 \phi} \sum_{j \in [r]} \delta_j + \frac{2R}{\psi_\ell}.\end{aligned}$$

The lemma follows by adding the initial value of $\|\boldsymbol{r}_{U_0,\ell}\|_1 \leq R$ to the above quantity. □

Now, we consider the more general case where A_i may be non-empty. Moreover, we would like to derive a bound in terms only of external cuts. Let δ_{ext} be the sum of δ_i's for which S_i is an external cut. Let $\Delta \coloneqq \Delta_1 + \cdots + \Delta_r$. We derive a bound when the following conditions are met.

Condition A.7.18. The following holds.

(i) Each internal cut S_i satisfies $\delta_i \le \frac{\phi\psi_\ell^2}{128}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)$.

(ii) Each cut S_i satisfies either $\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \le \frac{1}{4}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ or $\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \le \frac{1}{2}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$ when it is found.

(iii) $\psi_\ell \le \frac{1}{16}$, $R, \Delta \le \frac{\psi_\ell}{64}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$, and $\delta_{\mathrm{ext}} \le \frac{\phi\psi_\ell^2}{800}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$.

Observe that in either case of (ii), we have by (iii) that

$$\begin{aligned}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_i) &\le \mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) + \Delta \\ &\le \max\left\{\frac{1}{4}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0), \frac{1}{2}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})\right\} + \Delta \\ &\le \frac{5}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0). \end{aligned} \tag{A.17}$$

We will later show that Condition A.7.18 indeed holds (with high probability) throughout our algorithm for expander decomposition maintenance. For now we assume this is the case and prove the following Lemma A.7.20 using Lemma A.7.19 whose proof is deferred to Section A.10.

Lemma A.7.19. *Given a graph $G = (V, E)$ and a sequence of cuts $S_1, \ldots, S_k$ where $S_i \subseteq V_{i-1}$ with $V_i \coloneqq V_{i-1} \setminus S_i$ and $V_0 \coloneqq V$ satisfies*

$$\sum_{i\in[k]} \min\left\{\boldsymbol{c}_G(E_{G[V_{i-1}]}(S_i, \overline{S_i})), \boldsymbol{c}_G(E_{G[V_{i-1}]}(\overline{S_i}, S_i))\right\} < \phi \cdot \sum_{i\in[k]} \mathrm{vol}_{F,\boldsymbol{c}_G}(S_i) \tag{A.18}$$

and

$$\sum_{i\in[k]} \mathrm{vol}_{F,\boldsymbol{c}_G}(S_i) \le \alpha \cdot \mathrm{vol}_{F,\boldsymbol{c}_G}(V),$$

there is a $\left(\min\left\{\frac{\alpha}{2}, 1-\alpha\right\}\mathrm{vol}_{F,\boldsymbol{c}_G}(V)\right)$-balanced $\left(2\phi\min\left\{1, \frac{\alpha}{1-\alpha}\right\}\right)$-sparse cut in $(G, \boldsymbol{c})$ with respect to F.

Essentially, the above lemma says that if one can successively carve out many "small" sparse cuts S_i, then the original graph must have contained a "large" sparse cut. Now, in the following lemma, we establish that most of the change in $\boldsymbol{r}$ and vol_F comes from the external cuts.

Lemma A.7.20. *In Scenario A.7.13, if Condition A.7.18 holds, then we have $\|\boldsymbol{r}_{W_{U_r,\ell}}\|_1 \le \Gamma$ and $\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_r) \ge \mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0) - \Gamma$ for $\Gamma \coloneqq \frac{4(R+\Delta)}{\psi_\ell} + \frac{8}{\psi_\ell^2\phi}\delta_{\mathrm{ext}}$.*

Proof. Observe that we may imagine there is 0-th update with an empty cut S_0 with $A_0 \coloneqq A_1 \cup \cdots \cup A_r$, and after running UpdateWitness(U, S_0, A_0) we have $\|\boldsymbol{r}_{U,\ell}\|_1 \le R + \Delta$ and $W_{U,\ell}$ being a witness of $(G[U], \boldsymbol{c}_G, F_r)$ at which point we start considering Scenario A.7.13 with $A_i = \emptyset$ for all $i \in [r]$. Note that Condition A.7.18(i)

still holds in this case as well as (A.17). Also note that $\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0) \le 2\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ by the bound on Δ and thus Condition A.7.18(iii) implies $R, \Delta \le \frac{\psi_\ell}{32}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$ and $\delta_{\mathrm{ext}} \le \frac{\phi\psi_\ell^2}{400}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$.

Let $B_i := \mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0) - \mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_i) = \sum_{j\in[i]}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_j)$ be the total volume of the first i cuts. We first show that $B_r \le \frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$ must hold under the input assumption. Otherwise, let i be such that $B_i \le \frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$ and $B_{i+1} > \frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$. By $\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_{i+1}) \le \frac{5}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$ with (A.17) we know $B_{i+1} \le \frac{3}{4}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$. Let S_{int} and S_{ext} be the union of internal and external cuts among the first $i+1$ cuts. We have

$$\sum_{j\in[i+1]}\delta_j \le \frac{\phi\psi_\ell^2}{128}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_{\mathrm{int}}) + \delta_{\mathrm{ext}} \le \frac{\phi\psi_\ell^2}{12}B_{i+1}$$

by Condition A.7.18(i) and that (1) $\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_{\mathrm{int}}) \le B_{i+1}$, (2) $B_{i+1} \ge \frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$, and (3) $\delta_{\mathrm{ext}} \le \frac{\phi\psi_\ell^2}{400}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$. Because $B_{i+1} \le \frac{3}{4}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$, Lemma A.7.19 with $\alpha \le 3/4$ implies there is a $\left(\frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)\right)$-balanced $\frac{\phi\psi_\ell^2}{2}$-sparse cut in $(G[U_0], \boldsymbol{c}_G)$ with respect to F_r. This is a contradiction to Claim A.7.7 with the fact that $\frac{2(R+\Delta)}{\psi_\ell} \le \frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$.

As a result, we may assume $B_r \le \frac{1}{8}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$. If $B_r \le \frac{4(R+T)}{\psi_\ell}$, then we have $\sum_{j\in[r]}\delta_j \le \frac{4(R+T)}{\psi_\ell}\cdot\frac{\phi\psi_\ell^2}{128} + \delta_{\mathrm{ext}}$ and the lemma follows by applying Lemma A.7.16. Otherwise, letting S'_{int} be the union of all internal cuts, if $\delta_{\mathrm{ext}} \le \frac{31\phi\psi_\ell^2}{128}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S'_{\mathrm{int}})$ then we must have

$$\sum_{j\in[r]}\delta_j \le \frac{\phi\psi_\ell^2}{128}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S'_{\mathrm{int}}) + \delta_{\mathrm{ext}} \le \frac{\phi\psi_\ell^2}{4}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S'_{\mathrm{int}}) \le \frac{\phi\psi_\ell^2}{4}B_r,$$

which again by Lemma A.7.19 with $\alpha \le 1/4$ implies the existence of a $\frac{B_r}{2}$-balanced $\frac{\phi\psi_\ell^2}{2}$-sparse cut which contradicts Claim A.7.7. To this end, we have shown that $\delta_{\mathrm{ext}} \ge \frac{31\phi\psi_\ell^2}{128}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S'_{\mathrm{int}})$ and therefore $\sum_{j\in[r]}\delta_j \le 2\delta_{\mathrm{ext}}$. We can now apply Lemma A.7.16 to conclude bound on $\|\boldsymbol{r}_{W_r,\ell}\|_1$.

As for the bound on $\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_r)$, by the discussion above if $B_r > \frac{4(R+T)}{\psi_\ell}$ then $\sum_{j\in[r]}\delta_j \le 2\delta_{\mathrm{ext}}$. Since $B_r \le \frac{1}{4}\mathrm{vol}_{F_r,\boldsymbol{c}_G}(U_0)$ this means that we must have $2\delta_{\mathrm{ext}} \ge \frac{\phi\psi_\ell^2}{4}B_r$, otherwise the same argument above implies the existence of $\frac{B_r}{2}$-balanced $\frac{\phi\psi_\ell^2}{2}$-sparse cut that contradicts Claim A.7.7. This thus leaves us with $B_r \le \frac{8}{\phi\psi_\ell^2}\delta_{\mathrm{ext}}$ which the lemma statement asserts.

□

A.7.4 Maintaining Expander Decomposition

We now present the algorithm for maintaining expander decomposition while interacting with $\mathcal{M}_{\mathrm{prev}}$ which in turn gives the algorithm for converting from k-level

hierarchy to a $(k+1)$-level one with significantly fewer cut edges. The overall structure of our algorithms is similar to [HKPW23, Algorithm 3], and we use the stability properties established earlier to derive an expected worst-case recourse guarantee.

Setup. Given the input $(k, \alpha_{\text{prev}}, \beta_{\text{prev}}, \phi_{\text{prev}}, T_{\text{prev}})$-hierarchy maintainer $\mathcal{M}_{\text{prev}}$, to maintain the $(k+1)$-th level (and thereby getting a better maintainer $\mathcal{M}$), we will maintain the graph $G_{\mathcal{M}}$ in which F is ϕ-expanding and its collection of strongly connected components $\mathcal{U}$. We start with F being the edge set not handled by $\mathcal{M}_{\text{prev}}$. For each $U \in \mathcal{U}$ we will maintain an estimate τ_U of $\text{vol}_{F,\boldsymbol{c}_G}(U)$, a vector $\boldsymbol{\gamma}_U \in \mathbb{N}^U$, and $L+1$ witnesses $(W_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell} \to G[U]})$ for $\ell \in \{0, \ldots, L\}$, where $(W_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell} \to G[U]})$ is an $(\infty, \phi, \psi_\ell)$-witness of $(G[U], \boldsymbol{c}_G, F)$ with respect to $\boldsymbol{\gamma}_U$, with parameters

$$\psi_L \coloneqq \widetilde{\psi} \quad \text{and} \quad \psi_\ell \coloneqq \frac{1}{2} \cdot \left(\frac{\psi_{\ell+1}^2}{2048 \cdot c_{\text{A.6.1}} \cdot z^{3L}} \right)^L \tag{A.19}$$

that satisfy

$$\frac{1}{\psi_0} \leq \log n^{L^{O(L)}} \quad \text{and} \quad \frac{\psi_{\ell+1}^2}{\psi_\ell^{1/L}} \geq \Omega(\log^{3L} n). \tag{A.20}$$

The algorithm will ensure that, with high probability, each $\|\boldsymbol{r}_{W,\ell}\|_1$ falls between $\frac{\psi_\ell}{10} \tau_U^{\ell/L}$ and $\tau_U^{\ell/L}$. This can be enforced deterministically by rebuilding a witness whenever $\|\boldsymbol{r}_{W,\ell}\|_1$ grows too large. However, that leaves us with only an amortized guarantee which as we have argued in Section A.7.1 does not suffice for our purposes. To achieve a stronger expected worst-case recourse, we define the following distribution $\mathcal{R}_{t,\tau}$ on $\{0, \ldots, L\}$ from which we will sample a random level to rebuild after every update:

$$\Pr_{x \sim \mathcal{R}_{t,\tau}}[x \geq \ell] \coloneqq \min\left\{1, \frac{t}{\psi_0^2 \cdot \tau^{\ell/L}} \cdot c_{\text{rb}} \ln n\right\} \tag{A.21}$$

where c_{rb} is a fixed constant that controls the exponents in the with-high-probability statements that we will establish later.

Algorithm A.4 is the implementation of the Init() and Cut() subroutines which given input parameters $L \in \mathbb{N}$ and $\phi \in (0,1)$ converts $\mathcal{M}_{\text{prev}}$ into a $(k+1, \alpha, \beta, \phi', T)$-hierarchy maintainer $\mathcal{M}$ for parameters α, β, ϕ', T that we will establish in the end of the section. These subroutines rely on the internal subroutine MaintainExpander(U, ℓ) whose implementation is given in Algorithm A.5.

We say that $W_{U,\ell}$ is *rebuilt* if MaintainExpander(U, ℓ) enters Line 23. Recall that one of our goals is to ensure that $\|\boldsymbol{r}_{U,\ell}\|_1$ falls between $\frac{\psi_\ell}{10} \tau_U^{\ell/L}$ and $\tau_U^{\ell/L}$, and we show that this is indeed the case right after $W_{U,\ell}$ is rebuilt.

Lemma A.7.21. *If $W_{U,\ell}$ is rebuilt at Line 23, then the new $(W_{U,\ell}, \boldsymbol{c}_{U,\ell}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell} \to G[U]})$ is a $\left(\frac{\psi_\ell}{10} \tau_U^{\ell/L}, \phi, \psi_\ell\right)$-witness of $(G[U], \boldsymbol{c}_G, F)$.*

Algorithm A.4: Implementation of $(k+1, \alpha, \beta, \phi', T)$-hierarchy maintainer

global : parameters $L \in \mathbb{N}$ and $\phi \in (0,1)$
global : the graph $G_{\mathcal{M}}$ maintained by $\mathcal{M}$
global : the collection $\mathcal{U}$ of SCCs of $G_{\mathcal{M}}$
global : the terminal edge set F

function INIT$(G, \boldsymbol{c}_G)$
  Initialize $F \leftarrow \mathcal{M}_{\text{prev}}.\text{INIT}(G, \boldsymbol{c}_G)$.
  Initialize $G_{\mathcal{M}} \leftarrow G$ and $\mathcal{U} \leftarrow \{V\}$.
  Let $X \leftarrow$ MAINTAINEXPANDER$(V,\ L)$.
  Run POSTPROCESS(V) and then **return** X.
function CUT(D)
  $G_{\mathcal{M}} \leftarrow G_{\mathcal{M}} \setminus D$.
  $X \leftarrow \emptyset$ and $Q \leftarrow \emptyset$.
  Let $\mathcal{U}_D \leftarrow \{U \in \mathcal{U} : D \cap G[U] \neq \emptyset\}$ and $U_D \coloneqq \bigcup_{U \in \mathcal{U}_D} U$
  for $U \in \mathcal{U}_D$ **do**
    `// see input requirement of Cut()`
    Let $D_U \coloneqq D \cap G[U]$ and $S_U \subseteq U$ be such that
     $\text{vol}_{F,\boldsymbol{c}_G}(S_U) \leq \text{vol}_{F,\boldsymbol{c}_G}(U \setminus S_U)$ and either $E_{G[U]}(S_U, \overline{S_U})$ or
     $E_{G[U]}(\overline{S_U}, S_U)$ equals D_U.
    `// UpdateWitness() remove` S_U `from` U `and add` A_U `to` F
    Run $A_U \leftarrow \mathcal{M}_{\text{prev}}.\text{CUT}(D_U)$ and UPDATEWITNESS(U, S_U, A_U).
    Sample $k \sim \mathcal{R}_{\boldsymbol{c}_G(D_U)/\phi + 2\boldsymbol{c}_G(A_U), \tau_U}$ and
     $X \leftarrow X \cup$ MAINTAINEXPANDER$(U,\ k)$.
    Add S_U to Q.
  for $S \in Q$ **do** $X \leftarrow X \cup$ MAINTAINEXPANDER$(S,\ L)$.
  Run POSTPROCESS(U_D) and then **return** X.
  return X.
function POSTPROCESS(Y)
  `// ensure` $\mathcal{U}$ `is exactly the SCCs of` $G_{\mathcal{M}}$
  for $U \in \mathcal{U}$ *such that* $U \subseteq Y$ **do**
    Let $U_1, \ldots, U_k$ be the strongly connected components of $G[U]$.
    Replace U in $\mathcal{U}$ by $U_1, \ldots U_k$.

Proof. We have $\|\boldsymbol{r}_{U,\ell}\|_1 = \|\boldsymbol{r}_1\| + \|\boldsymbol{r}_2\| \leq \frac{\psi_\ell}{10} \cdot \tau_U^{\ell/L}$. Let $\psi'_\ell \coloneqq \frac{\psi_{\ell+1}^4}{2048 \cdot c_{A.6.1} \cdot z^2}$ as in Lemma A.7.10. We have $\psi_\ell \leq \frac{1}{2}{\psi'_\ell}^2$ by (A.19). We then have $\deg_{W_{U,\ell}, \boldsymbol{c}_{U,\ell}}(v) + \boldsymbol{r}_{U,\ell}(v) \leq \frac{2}{\psi'_\ell} \deg_{F,\boldsymbol{c}_G}(v) \leq \frac{1}{\psi_\ell} \deg_{F,\boldsymbol{c}_G}(v)$. Also, the congestion of $\Pi_{W_{U,\ell} \to G[U]}$ is bounded by $2 \cdot \frac{1}{\phi\psi'_\ell} \leq \frac{1}{\phi\psi_\ell}$. It thus remains to verify Property (3). Consider a cut where $\boldsymbol{\gamma}_U(S) \leq \boldsymbol{\gamma}_U(\overline{S})$. We have $\boldsymbol{c}_1(E_{W_1}(S, \overline{S})) + \boldsymbol{r}_1(S) \geq \psi'_\ell(\text{vol}_{W_1, \boldsymbol{c}_1}(S) + \boldsymbol{r}_1(S))$.

Algorithm A.5: Maintaining expander decomposition

global : a hierarchy $\mathcal{H}_{\text{prev}}$ of $G_{\mathcal{M}} \setminus F$ maintained by $\mathcal{M}_{\text{prev}}$

function MaintainExpander(U, ℓ)
  if $\text{vol}_{F,\boldsymbol{c}_G}(U) < 1/\phi$ **then return** $\emptyset$.
  $X \leftarrow \emptyset$ and $Q \leftarrow \emptyset$.
  loop
    if $\ell = L$ **then**
      $\tau_U \leftarrow \frac{\psi_0^2}{64z}\text{vol}_{F,\boldsymbol{c}_G}(U)$.
      Run procedure CutOrEmbed($G[U], \boldsymbol{c}_G, F, \phi, \frac{\psi_L}{10}\tau_U, \mathcal{H}_{\text{prev}}[U]$).
    else
      Run procedures PruneOrRepair($G[U], \boldsymbol{c}_G, F, \boldsymbol{r}_{U,\ell+1}, W_{U,\ell+1}, \Pi_{W_{U,\ell+1}\to G[U]}, \phi, \psi_{\ell+1}, \frac{\psi_\ell}{20} \cdot \tau_U^{\ell/L}, \mathcal{H}_{\text{prev}}[U]$) and PruneOrRepair($\overleftarrow{G}[U], \boldsymbol{c}_G, F, \boldsymbol{r}_{U,\ell+1}, \overleftarrow{W}_{U,\ell+1}, \Pi_{\overleftarrow{W}_{U,\ell+1}\to\overleftarrow{G}[U]}, \phi, \psi_{\ell+1}, \frac{\psi_\ell}{20} \cdot \tau_U^{\ell/L}, \mathcal{H}_{\text{prev}}[U]$).
    if *a cut S is returned* **then**
      Let D be edge sets among $E_{G[U]}(S, \overline{S})$ and $E_{G[U]}(\overline{S}, S)$ with smaller total capacities.
      update $X \leftarrow X \cup D$ and $G_{\mathcal{M}} \leftarrow G_{\mathcal{M}} \setminus D$
      Let $A \leftarrow \mathcal{M}_{\text{prev}}.\text{Cut}(D)$ and run UpdateWitness(U, S, A).
      `// assert` $A \subseteq U$
      Add S to Q.
      if *a sample $k \sim \mathcal{R}_{2\boldsymbol{c}_G(A),\tau_U}$ satisfies $k > \ell$* **then**
        Update $X \leftarrow X \cup$ MaintainExpander(U, k).
        break.
    else
      if $\ell = L$ **then**
        Set $(W_{U,L}, \boldsymbol{c}_{U,L}, \boldsymbol{r}_{U,\ell}, \Pi_{W_{U,\ell}\to G[U]})$ to be the witness returned by CutOrEmbed(), and $\boldsymbol{\gamma}_U \leftarrow \boldsymbol{\gamma}$.
      else
        Let $(W_1, \boldsymbol{c}_1, \boldsymbol{r}_1, \Pi_{W_1\to G[U]})$ and $(W_2, \boldsymbol{c}_2, \boldsymbol{r}_2, \Pi_{W_2\to\overleftarrow{G}[U]})$ be the witnesses returned by PruneOrRepair().
        Set $W_{U,\ell} \leftarrow W_1 \cup \overleftarrow{W_2}$, $\boldsymbol{c}_{U,\ell} \leftarrow \boldsymbol{c}_1 + \boldsymbol{c}_2$, and $\boldsymbol{r}_{U,\ell} \leftarrow \boldsymbol{r}_1 + \boldsymbol{r}_2$.
        `// the witness` $W_{U,\ell}$ `is rebuilt`
      break.
  if $\ell > 0$ **then** $X \leftarrow X \cup$ MaintainExpander($U, \ell - 1$).
  for $S \in Q$ **do** $X \leftarrow X \cup$ MaintainExpander(S, L).
  return X.

Note that due to Property (2), it holds that $\psi'_\ell(\deg_{W_2,\boldsymbol{c}_2}(v)+\boldsymbol{r}_2(v)) \le \deg_{W_1,\boldsymbol{c}_1}(v)+\boldsymbol{r}_1(v) \le \frac{1}{\psi'_\ell}(\deg_{W_2,\boldsymbol{c}_2}(v)+\boldsymbol{r}_2(v))$. Consequently, we have

$$\begin{aligned}\boldsymbol{c}_{U,\ell}(E_{W_{U,\ell}}(S,\overline{S})) &\ge \boldsymbol{c}_1(E_{W_1}(S,\overline{S}))\\ &\ge \psi'_\ell(\mathrm{vol}_{W_1,\boldsymbol{c}_1}(S)+\boldsymbol{r}_1(S))\\ &\ge \frac{{\psi'_\ell}^2}{2}(\mathrm{vol}_{W_{U,\ell},\boldsymbol{c}_{U,\ell}}(S)+\boldsymbol{r}_{U,\ell}(S)).\end{aligned}$$

Similarly, if $\boldsymbol{\gamma}_U(S) > \boldsymbol{\gamma}_U(\overline{S})$, then

$$\begin{aligned}\boldsymbol{c}_{U,\ell}(E_{W_{U,\ell}}(S,\overline{S})) &\ge \boldsymbol{c}_2(E_{\overleftarrow{W_2}}(S,\overline{S})) = \boldsymbol{c}_2(E_{W_2}(\overline{S},S))\\ &\ge \psi'_\ell(\mathrm{vol}_{W_1,\boldsymbol{c}_1}(S)+\boldsymbol{r}_1(S)) \ge \frac{{\psi'_\ell}^2}{2}(\mathrm{vol}_{W_{U,\ell},\boldsymbol{c}_{U,\ell}}(S)+\boldsymbol{r}_{U,\ell}(S)).\end{aligned}$$

This concludes the proof. □

As we have a bound on how large $\boldsymbol{r}_{U,\ell}(v)$ can be, the bound of Lemma A.7.10 with respect to $\mathrm{vol}_F(S)+\boldsymbol{r}_{U,\ell}(S)$ can be used to bound the actual volume.

Claim A.7.22. *Each cut S found at Line 9 satisfies (1) $\min\{\boldsymbol{c}_G(E_{G[U]}(S,\overline{S})), \boldsymbol{c}_G(E_{G[U]}(\overline{S},S))\} \le \frac{\phi\psi_{\ell+1}^2}{128}\mathrm{vol}_{F,\boldsymbol{c}_G}(S)$, and (2) $\mathrm{vol}_{F,\boldsymbol{c}_G}(S) \ge \frac{\psi_{\ell+1}^2\psi_\ell}{640z}\tau_U^{\ell/L}$.*

Proof. The cut S found by Lemma A.7.10 satisfies

$$\min\{\boldsymbol{c}_G(E_{G[U]}(S,\overline{S})), \boldsymbol{c}_G(E_{G[U]}(\overline{S},S))\} \le \frac{\phi\psi_{\ell+1}^3}{256}(\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}_{U,\ell+1}(S))$$

and $\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}_{U,\ell+1}(S) \ge \frac{\psi_{\ell+1}}{16z}\cdot\frac{\psi_\ell}{20}\tau_U^{\ell/L}$. By Definition A.7.6(2), we have $\boldsymbol{r}_{U,\ell+1}(S) \le \frac{1}{\psi_{\ell+1}}\mathrm{vol}_{F,\boldsymbol{c}_G}(S)$, and therefore $\mathrm{vol}_{F,\boldsymbol{c}_G}(S)+\boldsymbol{r}_{U,\ell+1}(S) \le \frac{2}{\psi_{\ell+1}}\mathrm{vol}_{F,\boldsymbol{c}_G}(S)$. The claim follows. □

Recall that a cut passed to UpdateWitness() is *internal* if it comes from Line 13 in Algorithm A.5 and *external* if it comes from Line 12 in Algorithm A.4. We further call such an internal cut *level-*$(\ell+1)$ as it is found based on $W_{U,\ell+1}$ when running MaintainExpander(X,ℓ).

Definition A.7.23. A witness $W_{U,\ell}$ is *valid* if since the last time it was rebuilt, we only call RemoveCut(U,S,A) on it with either external or level-ℓ' internal cuts with $\ell' \le \ell$; otherwise, $W_{U,\ell}$ is *invalid*.

A witness $W_{U,\ell}$ is *being rebuilt* from the moment the algorithm enters MaintainExpander(X,k) for some $k \ge \ell$ until either it is actually rebuilt or the call to MaintainExpander(X,k) returns. By definition, if $W_{U,\ell}$ is currently being rebuilt, then so are all $W_{U,\ell'}$ with $\ell' < \ell$. Observe that a witness is always valid

unless either it is currently being rebuilt or has volume $\mathrm{vol}_{F,\boldsymbol{c}_G}(U) < 1/\phi$ (due to the early return on Line 2).[30] Let us call such a U with $\mathrm{vol}_{F,\boldsymbol{c}_G}(U) < 1/\phi$ *negligible*.

Observation A.7.24. *Each witness $W_{U,\ell}$ for a non-negligible U remains valid unless it is currently being rebuilt, in which case it must actually be rebuilt before the call to the corresponding* MaintainExpander(U,k) *returns. Moreover, if the algorithm is currently running* MaintainExpander(U,ℓ)*, then all $W_{U,k}$ where $k > \ell$ are valid.*

Observation A.7.25. *If U becomes negligible at some point, then it remains negligible afterward.*

Consider two timestamps $t_1 < t_2$ throughout the execution of the algorithm.

Definition A.7.26. A tuple (U,ℓ,t_1,t_2) is *active* if (i) U is non-negligible at time t_2 (and thus from t_1 to t_2 by Observation A.7.25) and (ii) $W_{U,\ell}$ is not rebuilt nor being rebuilt in any point of time between t_1 and t_2 (inclusively).

Consider an active tuple (U,ℓ,t_1,t_2). Let $\delta^{(U,\ell)}_{\mathrm{ext}}(t_1,t_2)$ be the sum of the capacities of D_U's of the Cut(D) calls that happen from time t_1 to t_2. Likewise, let $\Delta^{(U,\ell)}(t_1,t_2)$ be *two times* the sum of the capacities of A's of the UpdateWitness(U,S,A) calls happened in this period of time. In other words, $\Delta^{(U,\ell)}$ is an upper bound on the units of volume increased in U from t_1 to t_2. We show that for an active tuple (U,ℓ,t_1,t_2), with high probability both $\delta^{(U,\ell)}_{\mathrm{ext}}(t_1,t_2)$ and $\Delta^{(U,\ell)}(t_1,t_2)$ are bounded. Note that as the value of τ_U only changes in MaintainExpander(U,L), during this period of time τ_U remains unchanged (otherwise $W_{U,\ell}$ would have been rebuilt at some point in between). Also note that it is important to establish the bound against *any* (possibly adversarial) sequence of Cut() calls, since as we have seen earlier we will encapsulate this hierarchy maintainer into another one that has a better quality. The update sequence we see now thus depends on our previous outputs (hence the previous randomness used).

Lemma A.7.27. *For any (possibly adaptive adversarial) sequence of* Cut() *and any active tuple (U,ℓ,t_1,t_2), with high probability, we have $\Delta^{(U,\ell)}(t_1,t_2) \le \frac{\psi_\ell}{10}\tau_U^{\ell/L}$ and $\delta^{(U,\ell)}_{\mathrm{ext}}(t_1,t_2) \le \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/U}$.* [31]

Proof. Let Δ_i be two times the total capacities of A in the i-th call of RemoveCut(U,S,A) from time t_1 to t_2. If $\Delta^{(U,\ell)}(t_1,t_2) > \frac{\psi_\ell}{10}\tau_U^{\ell/L}$, then the probability that

[30] Note that this would have been vacuously true if our algorithm does not have the early break on Line 17 in Algorithm A.5 (like in [HKPW23, Algorithm 3]). Still, if we break early on Line 17, then the call to MaintainExpander(X,k) on Line 16 will be in charge of rebuilding $W_{U,\ell}$.

[31] The exponent in the with high probability statement depends on our choice of the constant c_{rb} in (A.21).

none these subroutines called MaintainExpander(U,k) for some $k \geq \ell$ (which scheduled $W_{U,\ell}$ to be rebuilt) is at most

$$\prod_i \left(1 - \frac{\Delta_i}{\psi_0^2 \tau_U^{\ell/L}} \cdot c_{\mathrm{rb}} \ln n\right) \leq \exp\left(-\sum_i \Delta_i \cdot \frac{c_{\mathrm{rb}} \ln n}{\psi_0^2 \tau_U^{\ell/L}}\right) \leq \exp\left(-\frac{c_{\mathrm{rb}} \ln n}{10\psi_0}\right) \leq n^{-c_{\mathrm{rb}}/10}.$$

Similarly, let D_i be the total capacities $\boldsymbol{c}_G(D_U)$ of D_U in the i-th call to Cut(D) from time t_1 to t_2. If $\delta_{\mathrm{ext}}^{(U,\ell)}(t_1,t_2) = \sum_i D_i > \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/U}$, then the probability that none of the Cut() called MaintainExpander(U,k) for some $k \geq \ell$ is at most

$$\prod_i \left(1 - \frac{D_i}{\phi} \cdot \frac{1}{\psi_0^2 \tau_U^{\ell/L}} \cdot c_{\mathrm{rb}} \ln n\right) \leq \exp\left(-\sum_i \frac{D_i}{\phi} \cdot \frac{c_{\mathrm{rb}} \ln n}{\psi_0^2 \tau_U^{\ell/L}}\right) \leq \exp\left(-\frac{c_{\mathrm{rb}} \ln n}{40}\right) = n^{-c_{\mathrm{rb}}/40}.$$

Consequently, with high probability, the bounds of $\Delta^{(U,\ell)}(t_1,t_2) \leq \frac{\psi_\ell}{10}\tau_U^{\ell/L}$ and $\delta_{\mathrm{ext}}^{(U,\ell)}(t) \leq \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/L}$ hold if $W_{U,\ell}$ is not currently scheduled to be rebuilt. Observe that we generate new randomness for each of our random choices, and thus this high probability guarantee works against any update sequence. □

In light of Lemma A.7.27, let $\mathcal{K}$ be the event such that

$$\Delta^{(U,\ell)}(t_1,t_2) \leq \frac{\psi_\ell}{10}\tau_U^{\ell/L} \quad \text{and} \quad \delta_{\mathrm{ext}}^{(U,\ell)}(t_1,t_2) \leq \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/L} \tag{A.22}$$

hold for all active tuples (U,ℓ,t_1,t_2). Observe that there are only $\mathrm{poly}(n)$ effective timestamps throughout any possible execution of the algorithm[32] and for each effective timestamp there are only $O(n)$ possible U's at this time since they are vertex-disjoint, and thus by Lemma A.7.27 and a union bound $\mathcal{K}$ happens with high probability.

Lemma A.7.28. *Conditioned on $\mathcal{K}$, for any $U \in \mathcal{U}$ and time t, if $W_{U,L}$ is not currently being rebuilt at time t, then we have $\frac{\psi_0^2}{128z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U) \leq \tau_U \leq \frac{\psi_0^2}{32z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U)$; moreover, and for each $\ell \in \{0,\ldots,L\}$, if $W_{U,\ell}$ is not currently being rebuilt at time t, then $\|\boldsymbol{r}_{U,\ell}\|_1 \leq \tau_U^{\ell/L}$ holds.*

Proof. We prove the statement by an induction on the time t. Fix a $U \in \mathcal{U}$ and $\ell \in \{0,\ldots,L\}$ for which $W_{U,\ell}$ is not currently being rebuilt. Consider the last time $t_{\mathrm{last}}^{(U,\ell)} < t$ that it was rebuilt. Let U_0 and F_0 be the set U and F at time $t_{\mathrm{last}}^{(U,\ell)}$. By the inductive hypothesis at time $t_{\mathrm{last}}^{(U,\ell)}$, we have $\tau_U \leq \frac{\psi_0}{32z}\mathrm{vol}_{F_0}(U_0)$. Note that the value of τ_U remains unchanged from $t_{\mathrm{last}}^{(U,\ell)}$ to t, otherwise $W_{U,\ell}$ would have currently been being rebuilt. By Lemma A.7.21, we have $\|\boldsymbol{r}_{U_0,\ell}\|_1 \leq \frac{\psi_\ell}{10}\tau_U^{\ell/L}$. Observe that $(U,\ell,t_{\mathrm{last}}^{(U,\ell)},t)$ is active, and thus what happens from $t_{\mathrm{last}}^{(U,\ell)}$ to t is

[32] The subroutine UpdateWitness() will be called at most n times and $F \subseteq E$ always hold so there will be at most $O(m)$ terminal additions.

modeled by Scenario A.7.13. Our goal is thus to apply Lemma A.7.20 on $W_{U,\ell}$ (with $R \le \frac{\psi_\ell}{10}\tau_U^{\ell/L}$) to bound the increase in $\|\boldsymbol{r}_{U,\ell}\|_1$. For that we need to verify that Condition A.7.18 holds. In the remainder of the proof we adapt the notation in Scenario A.7.13 (e.g., U_i, S_i, and F_i).

Condition A.7.18(i). Note that by Observation A.7.24, $W_{U,\ell}$ is currently valid, meaning that all the internal cuts S_i for which UpdateWitness$(U, S_i, \cdot)$ is called are of level-ℓ' for $\ell' < \ell$. By Claim A.7.22, each such cut S_i satisfies

$$\begin{aligned}\min\{\boldsymbol{c}_G(E_{G[U_{i-1}]}(S_i, \overline{S_i})), \boldsymbol{c}_G(E_{G[U_{i-1}]}(\overline{S_i}, S_i))\} &\le \frac{\phi\psi_{\ell'+1}^2}{128}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)\\ &\le \frac{\phi\psi_\ell^2}{128}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i).\end{aligned}$$

Condition A.7.18(ii). For external cuts, since we always let S_U be the side with smaller volume on Line 12 in Algorithm A.4, we indeed have $\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \le \frac{1}{2}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$. This is the same for internal cuts of level-L by Lemma A.7.11. For $\ell < L$, by Lemma A.7.10, every such cut S_i satisfies $\mathrm{vol}_{F_i,\boldsymbol{c}_G}(S_i) + \boldsymbol{r}_{U,\ell'+1}(S) \le \frac{8z}{\psi_{\ell'+1}}R$ for some $\ell' < \ell$ if $(W_{U,\ell'+1}, \boldsymbol{r}_{U,\ell'+1}, \Pi_{W_{U,\ell'+1}\to G[U]})$ is an R-witness at the time t_i it was found. By the inductive hypothesis at time t_i, we have $R \le \tau_U^{(\ell'+1)/L} \le \tau_U \le \frac{\psi_0^2}{32z}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$, and thus we have $\mathrm{vol}_{F_i,\boldsymbol{c}_G}(S_i) \le \frac{1}{4}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$.

Condition A.7.18(iii). The fact that $\delta_\ell \le \frac{1}{16}$ is straightforward. That $R \le \frac{\psi_\ell}{64}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ is by Lemma A.7.21 which shows that right after the rebuild we have $R = \|\boldsymbol{r}_{U,\ell}\|_1 \le \frac{\psi_\ell}{10}\tau_U^{\ell/L}$. Applying the bound of $\tau_U \le \frac{\psi_0^2}{32z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U_0)$ establishes the fact. For the last two bounds, by the conditioning on $\mathcal{K}$, we have $\Delta^{(U,\ell)}(t_{\text{last}}^{(U,\ell)}, t) \le \frac{\psi_\ell}{10}\tau_U^{\ell/L}$ and $\delta_{\text{ext}}^{(U,\ell)}(t_{\text{last}}^{(U,\ell)}, t) \le \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/L}$. With $\tau_U \le \frac{\psi_0}{32z}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$, we additionally have $\Delta^{(U,\ell)}(t_{\text{last}}^{(U,\ell)}, t) \le \frac{\psi_\ell}{64}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ and $\delta_{\text{ext}}^{(U,\ell)}(t_{\text{last}}^{(U,\ell)}, t) \le \frac{\phi\psi_\ell^2}{800}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$.

Therefore, against any update sequence, with high probability Condition A.7.18 holds. Applying Lemma A.7.20 on $W_{U,\ell}$ (with $R \coloneqq \frac{\psi_\ell}{10}\tau_U^{\ell/L}$, $\Delta \coloneqq \Delta^{(U,\ell)}(t_{\text{last}}^{(U,\ell)}, t)$, and $\delta_{\text{ext}}^{(U,\ell)}(t_{\text{last}}^{(U,\ell)}, t)$), we conclude that $\|\boldsymbol{r}_{U,\ell}\|_1 \le \frac{4(R+\Delta)}{\psi_\ell} + \frac{8}{\psi_\ell^2\phi}\delta_{\text{ext}} \le \tau_U^{\ell/L}$.

For the bound on τ_U, we use our previous arguments when $\ell = L$. Notice that τ_U is set to $\frac{\psi_0^2}{64z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U)$ at time $t_{\text{last}}^{(U,L)}$ on Line 6 in Algorithm A.5. As we have argued above, Lemma A.7.20 applied on $W_{U,L}$ implies that the volume of the current U is at least $\frac{64z}{\psi_0}\tau_U - \left(\frac{4(R+\Delta)}{\psi_\ell} + \frac{8}{\psi_\ell^2\phi}\right) \ge \frac{64z}{\psi_0^2}\tau_U - \tau_U \ge \frac{32z}{\psi_0^2}\tau_U$ (where $R \le \frac{\psi_L}{10}\tau_U$, $\Delta \coloneqq \Delta^{(U,L)}(t_{\text{last}}^{(U,L)}, t)$, and $\delta_{\text{ext}} \coloneqq \delta_{\text{ext}}^{(U,L)}(t_{\text{last}}^{(U,L)}, t)$). This proves the upper bound of τ_U. On the other hand, the volume of U can increase by at most $\Delta^{(U,L)}(t_{\text{last}}^{(U,L)}, t) \le \frac{\psi_L}{10}\tau_U \le \frac{\psi_0}{10}\tau_U$. This shows that $\tau_U \ge \frac{\psi_0^2}{128z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U)$, which completes the proof of the lemma. □

To this end, we can now essentially conclude the correctness of our algorithm, except for the running time and output size guarantees that we will establish in Section A.7.5. Below we prove several useful properties that our algorithm satisfies.

Essential Properties of Algorithm. Let X_0 be the output of $\textsc{Init}(G)$ and let X_i be the output of $\textsc{Cut}(D_i)$ where D_i is the i-th update to $\mathcal{M}$. We first verify that the outputs of $\textsc{Init}(G)$ and $\textsc{Cut}(D)$ are consistent with the graph $G_{\mathcal{M}}$ that our algorithm maintains internally (see Definition A.7.3 for the requirements).

Observation A.7.29. *After the i-th call to $\textsc{Cut}()$, the graph $G_{\mathcal{M}}$ is equal to $G \setminus (X_0 \cup D_1 \cup X_1 \cup \cdots \cup D_i \cup X_i)$ with $\mathcal{U}$ being the collection of its strongly connected components.*

Observation A.7.30. *The graph $G_{\mathcal{M}}$ we maintain satisfies $G_{\mathcal{M}} \supseteq G_{\mathcal{M}_{\text{prev}}}$, and the terminal set F we maintain satisfies $F \supseteq E(G_{\mathcal{M}}) \setminus E(G_{\mathcal{M}_{\text{prev}}})$.*

Observation A.7.31. *Whenever we call Lemma A.7.11 on Line 7 or Lemma A.7.10 on Line 9 in Algorithm A.5, the hierarchy $\mathcal{H}_{\text{prev}}[U]$ is a ϕ_{prev}-expander hierarchy of the graph $G[U] \setminus F$.*

Lemma A.7.32. *Algorithms A.4 and A.5 maintain that F is $\frac{\phi\psi_0^2}{2}$-expanding in $(G_{\mathcal{M}}, \boldsymbol{c}_G)$ after each update ($\mathcal{M}.\textsc{Cut}(D)$) against an adaptive adversary.*

Proof. After each update, for each $U \in \mathcal{U}$ and $\ell \in \{0, \ldots, L\}$ we have that $W_{U,\ell}$ is not being rebuilt for all $\ell \in \{0, \ldots, L\}$. By Lemma A.7.28, with high probability, if $\text{vol}_{F,\boldsymbol{c}_G}(U) \geq 1/\phi$, then $\|\boldsymbol{r}_{U,0}\| \leq 1$ which by Claim A.7.9 implies F is $\frac{\phi\psi_0^2}{2}$-expanding in $(G[U], \boldsymbol{c}_G)$ (note that by Claim A.7.12 $W_{U,0}$ is indeed a valid witness of $(G[U], \boldsymbol{c}_G, F)$). On the other hand, if $\text{vol}_{F,\boldsymbol{c}_G}(U) < 1/\phi$, then F is also $\frac{\phi\psi_0^2}{2}$-expanding in $(G[U], \boldsymbol{c}_G)$ by Claim A.7.8. This completes the proof. □

Claim A.7.33. *On Line 13 in Algorithm A.5, the set D satisfies the input requirement of $\mathcal{M}_{\text{prev}}.\textsc{Cut}(D)$ (see Definition A.7.3), the call runs in $T_{\text{prev}}(|U|)$ time, and it returns an edge set $A \subseteq E(G[U])$.*

Proof. By Observation A.7.30, the graph $G_{\mathcal{M}}$ is always a supergraph of $G_{\mathcal{M}_{\text{prev}}}$ which means that the $\text{SCC}(G_{\mathcal{M}_{\text{prev}}})$ is a refinement of $\text{SCC}(G_{\mathcal{M}})$. Let $\mathcal{U}' \subseteq \text{SCC}(G_{\mathcal{M}_{\text{prev}}})$ be the collection of SCCs of $G_{\mathcal{M}_{\text{prev}}}$ contained in U, where we must have that $U' := \bigcup_{U_i \in \mathcal{U}'} U_i = U$. Since D is a cut $E_{G[U]}(S, U \setminus S)$ for some $S \subseteq U$, for each $U_i \in \mathcal{U}'$ if $D \cap G[U_i] \neq \emptyset$ it must be that $G \cap G[U_i] = E_{G[U_i]}(S_i, U_i \setminus S_i)$ for some S_i. Indeed, this will be the case if we let $S_i := S \cap U_i$. This shows that the input requirement of $\mathcal{M}_{\text{prev}}.\textsc{Cut}(D)$ is satisfied and also that the set U_D defined in Definition A.7.3 is a subset of U. Therefore, $\mathcal{M}_{\text{prev}}.\textsc{Cut}(D)$ runs in $T_{\text{prev}}(|U|)$ time and returns an edge set $A \subseteq \bigcup_{U_i \in \mathcal{U}'} G[U_i] \subseteq G[U]$. □

A.7.5 Bounding Expected Recourse and Running Time

It now remains to argue both the expected running time and the expected output size of the subroutines Init and Cut. As the outputs of both these subroutines come from the MaintainExpander algorithm, we focus on establishing the guarantees for this function.

Good vs Bad States of Algorithm. In Section A.7.4 we have shown that throughout the algorithm, with high probability, we have $\|\boldsymbol{r}_{U,\ell}\| \leq \tau_U^{\ell/L}$ for all non-negligible U that is not currently being rebuilt. Let us define this as a *good* state of the algorithm and consider the following generalizations of this notion.

Definition A.7.34. A state of the algorithm is *good* if for all non-negligible $U \in \mathcal{U}$ and $\ell \in \{0, \ldots, L\}$ for which $W_{U,\ell}$ is not currently being rebuilt it holds that $\|\boldsymbol{r}_{U,\ell}\| \leq \tau_U^{\ell/L}$; and if $W_{U,L}$ is not currently being rebuilt then $\frac{\psi_0^2}{128z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U) \leq \tau_U \leq \frac{\psi_0^2}{32z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U)$ holds. The state is *borderline* if $\|\boldsymbol{r}_{U,\ell}\| \leq \frac{10}{\psi_\ell}\tau_U^{\ell/L}$ holds for these U and ℓ instead; and if $W_{U,L}$ is not currently being rebuilt then $\frac{\psi_0^2}{128z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U) \leq \tau_U \leq \frac{\psi_0^2}{32z}\mathrm{vol}_{F,\boldsymbol{c}_G}(U)$ holds. The state is *bad* if it is not borderline.

The reason why we need Definition A.7.34 and in particular the definition of borderline states for our analysis is that, while Lemmas A.7.27 and A.7.28 showed that the algorithm is always in a good state with high probability, it is *not true* that if we start from *any* good state, then we always stay in good states with high probability. Indeed, we can be in a good state in which one of the witnesses has $\|\boldsymbol{r}_{U,\ell}\|_1$ being very close to $\tau_U^{\ell/L}$, yet there is a constant probability that we will not rebuild it in the next update (which leads us to a borderline state). Note that this is not contradictory to Lemmas A.7.27 and A.7.28 because the probability that we are in such good states is small. While it is not true that a good state remains good, we can generalize Lemmas A.7.27 and A.7.28 and show that a good state never reaches a *bad* state with high probability. We defer the proof of the following lemma to Section A.10 since it is essentially identical to that of Lemmas A.7.27 and A.7.28.

Lemma A.7.35. *Conditioned on the algorithm being in a good state at the current moment, with high probability, the algorithm will remain in borderline states until termination.*

For simplicity of exposition, we will work with the quantities $\textsc{Size}_\ell(b, \lambda)$ and $\textsc{Time}_\ell(b, \lambda)$ that intuitively stand for the size of the cut $\textsc{MaintainExpander}(U, \ell)$ will return respectively the running time of the call, where $b = \mathrm{vol}_{F,\boldsymbol{c}_G}(U)$ and $\lambda = |U|$. We formally define them as follows. Consider $b \in \{0, \ldots, m\}$, $\lambda \in \{0, \ldots, n\}$, and $\ell \in \{0, \ldots, L\}$. Let $\mathscr{B}_{b,\lambda,\ell}$ be the collection of all possible calls of $\textsc{MaintainExpander}(U, \ell)$ that start when the algorithm is in a *borderline* state with $\mathrm{vol}_F(U) \leq b$ and $|U| \leq \lambda$. For each run $r \in \mathscr{B}_{b,\lambda,\ell}$ there are two random variables

$S_r \in \{0, \dots, m\}$[33] and $T_r \in \{0, \dots, \text{poly}(n)\}$[34] which represent the capacities $\boldsymbol{c}_G(X)$ of X that this MaintainExpander(U, ℓ) call returns respectively the running time of it; the randomness comes from both the sampling of $k \sim \mathcal{R}_{t,\tau}$ (on Line 13 in Algorithm A.4 and Line 15 in Algorithm A.5) and the expected guarantee of $\mathcal{M}_{\text{prev}}$.Cut() (see Definition A.7.3). Let Ω be the space of randomness. Let $S_r(\omega)$ and $T_r(\omega)$ be the realization of S_r and T_r on randomness ω. Additionally, we let $\widetilde{S}_r$ and $\widetilde{T}_r$ be random variables that act almost the same as S_r and T_r do, except if for some randomness ω the run $r \in \mathscr{B}_{b,\lambda,\ell}$ reaches a *bad* state then we define $\widetilde{S}_r(\omega)$ and $\widetilde{T}_r(\omega)$ to both be realized to zero.

Definition A.7.36. We define $\textsc{Size}_\ell(b, \lambda)$ to be the random variable on range $\{0, \dots, m\}$ such that $\textsc{Size}_\ell(b, \lambda)(\omega) \coloneqq \max_{r \in \mathscr{B}_{b,\lambda,\ell}} \widetilde{S}_r(\omega)$ for all $\omega \in \Omega$. Likewise, $\textsc{Time}_\ell(b, \lambda)$ is a random variable on $\{0, \dots, \text{poly}(n)\}$ such that $\textsc{Time}_\ell(b, \lambda)(\omega) \coloneqq \max_{r \in \mathscr{B}_{b,\lambda,\ell}} \widetilde{T}_r(\omega)$ for all $\omega \in \Omega$.

Note that by definition, $\textsc{Size}_\ell(b, \lambda)$ and $\textsc{Time}_\ell(b, \lambda)$ are increasing in both b and λ. The reason why we define $\textsc{Size}_\ell(b, \lambda)$ and $\textsc{Time}_\ell(b, \lambda)$ in terms of the rather bizarre-looking $\widetilde{S}_r$ and $\widetilde{T}_r$ is to ensure that we can assume the algorithm to be in a borderline state when doing the calculation which significantly simplifies matters. To convert a bound on $\textsc{Size}_\ell(b, \lambda)$ and $\textsc{Time}_\ell(b, \lambda)$ to the expected guarantee of the $\textsc{Cut}(D)$ subroutine, we use Lemma A.7.35 to argue that the contribution of bad states to the actual expectation can be made as small as n^{-100} using the following fact since the probability of reaching those states from a good one is inverse polynomially small).

Fact A.7.37. *If Y is a random variables in $\{0, \dots, \text{poly}(n)\}$ and $\mathcal{E}$ is an event that happens with high probability, then $\mathbb{E}[Y] \le \mathbb{E}[Y \mid \mathcal{E}] + n^{-100}$.*

This together with Lemma A.7.35 implies the following.

Lemma A.7.38. *If the algorithm is in a good state right before running* MaintainExpander(U, ℓ)*, then the subroutine runs in expected $\mathbb{E}[\textsc{Time}_\ell(\text{vol}_{F,\boldsymbol{c}_G}(U), |U|)] + n^{-100}$ time and outputs an X of expected size at most $\mathbb{E}[\textsc{Size}_\ell(\text{vol}_{F,\boldsymbol{c}_G}(U), |U|)] + n^{-100}$.*

Again, we recall that by Lemma A.7.28, with high probability, the algorithm is always in a good state when running $\textsc{Cut}(D)$ even against an adaptive adversary. Consequently, it remains to bound $\mathbb{E}[\textsc{Size}_\ell(b, \lambda)]$ and $\mathbb{E}[\textsc{Time}_\ell(b, \lambda)]$.

An Overestimating Approach. To bound the expected value of $\textsc{Size}_\ell(b, \lambda)$ and $\textsc{Time}_\ell(b, \lambda)$, we consider a run of MaintainExpander(U, ℓ) and write down a recurrence that upper bounds them based on Algorithm A.5. We often deliberately

[33] Recall that $m \le n^4$ is the total capacities in the graph G.

[34] It can be checked that the running time of the algorithm *always* runs in polynomial time.

overestimate the expectation. More specifically, while we are in a borderline state and might reach a bad state in the recursion which makes the realization of both $\textsc{Size}_\ell(b,\lambda)$ and $\textsc{Time}_\ell(b,\lambda)$ be defined as zero, we compute their values *as if we are always in borderline states*. Note that since the values of these random variables are non-negative, doing this can only overestimate their true values. This greatly simplifies setting up the recurrence.

Setup. Consider the execution of $\textsc{MaintainExpander}(U,\ell)$ which finds cuts $S_1,\dots,S_r$ through either Lemma A.7.11 or Lemma A.7.10 until we exit the while-loop by either successfully constructing/repairing the witness $W_{U,\ell}$ or sampling a $k>\ell$ on Line 15. Observe that the final return set X can be written as the union of four parts $X^{(1)}\cup X^{(2)}\cup X^{(3)}\cup X^{(4)}$, where

- $X^{(1)}$ are edges added on Line 13,
- $X^{(2)}$ are edges found by recursively calling $\textsc{MaintainExpander}(U,k)$ on Line 16 (let us call this a *fixing* recursion),
- $X^{(3)}$ are edges found by recursively calling $\textsc{MaintainExpander}(U,\ell-1)$ on Line 25 (let us call this a *downward* recursion),
- $X^{(4)}$ are edges found by recursively calling $\textsc{MaintainExpander}(S,L)$ on Line 26 for all the cuts S (let us call this a *rebuilding* recursion).

The running time of $\textsc{MaintainExpander}(U,\ell)$ can be similarly split into what it takes to compute each of the $X^{(i)}$'s. We adapt the same notation as in Scenario A.7.13: We let F_i be the terminal set after finding S_i and running the $F\leftarrow F\cup A$, and in particular F_0 is the initial set F when $\textsc{MaintainExpander}(U,\ell)$ is called. Let $U_0\coloneqq U$ and $U_i\coloneqq U_{i-1}\setminus S_i$ and let $\Delta_i\le 2\boldsymbol{c}_G(F_i\setminus F_{i-1})$ be the units of volume added due to S_i. We first prove some basic properties regarding the volume of each U_i and S_i.

Lemma A.7.39. *For $\phi\le\frac{1}{4\beta_{\text{prev}}}$, we have*

(1) $\mathbb{E}[\mathrm{vol}_{F_i,\boldsymbol{c}_G}(U_i)\mid F_{i-1},U_{i-1}]\le\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$ *for all $i\in[r]$,*

(2) $\mathbb{E}[\mathrm{vol}_{F_i,\boldsymbol{c}_G}(U_i)]\le\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ *for all $i\in[r]$,*

(3) $\mathbb{E}[\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_1)+\cdots+\mathrm{vol}_{F_r,\boldsymbol{c}_G}(S_r)]\le(1+8\phi\beta_{\text{prev}})\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$, *and*

(4) $\mathbb{E}[\boldsymbol{c}_G(X^{(1)})]\le\phi\cdot\mathbb{E}[\mathrm{vol}_{F_0,\boldsymbol{c}_G}(S_1)+\cdots+\mathrm{vol}_{F_{r-1},\boldsymbol{c}_G}(S_r)]$.

Proof. Let D_i be the set D defined on Line 11 when S_i is found. By Lemma A.7.11 and Claim A.7.22, we have $\boldsymbol{c}_G(D_i)\le\phi\cdot\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)$. Let A_i be the set A obtained by $A\leftarrow\mathcal{M}_{\text{prev}}.\textsc{Cut}(D_i)$ after S_i is found. By the guarantee of $\mathcal{M}_{\text{prev}}$, we have

$\mathbb{E}[\boldsymbol{c}_G(A_i) \mid S_i] \le \beta_{\text{prev}} \cdot \boldsymbol{c}_G(D_i) \le \beta_{\text{prev}}\phi \cdot \text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)$. Since we are adding at most $2\boldsymbol{c}_G(A_i)$ units of volume to U_{i-1} in total, we have

$$\mathbb{E}[\text{vol}_{F_i,\boldsymbol{c}_G}(U_i) \mid U_{i-1}, F_{i-1}, S_i] \le \text{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1}) - \text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) + 2\beta_{\text{prev}}\phi \cdot \text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \tag{A.23}$$

and

$$\mathbb{E}[\text{vol}_{F_i,\boldsymbol{c}_G}(S_i) \mid U_{i-1}, F_{i-1}] \le \text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) + 2\beta_{\text{prev}}\phi \cdot \text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i). \tag{A.24}$$

As $2\beta_{\text{prev}}\phi \le 1/2$, we see that in expectation $\text{vol}_{F_i,\boldsymbol{c}_G}(U_i)$ is smaller than the volume $\text{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$. This proves (1), and (2) simply follows from (1) and the law of total expectation.

For (3), we add a $\left(\frac{1-2\beta_{\text{prev}}\phi}{1+2\beta_{\text{prev}}\phi}\right)$-multiple of (A.24) to (A.23) and get

$$\mathbb{E}\left[\left(\frac{1-2\beta_{\text{prev}}\phi}{1+2\beta_{\text{prev}}\phi}\right)\text{vol}_{F_i,\boldsymbol{c}_G}(S_i) + \text{vol}_{F_i,\boldsymbol{c}_G}(U_i) - \text{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1}) \,\middle|\, U_{i-1}, F_{i-1}, S_i\right] \le 0$$

which implies

$$\mathbb{E}\left[\sum_{i\in[r]} \text{vol}_{F_i,\boldsymbol{c}_G}(S_i)\right] \le \frac{1+2\beta_{\text{prev}}\phi}{1-2\beta_{\text{prev}}\phi} \cdot \text{vol}_{F_0,\boldsymbol{c}_G}(U_0) \le (1+8\beta\phi)\text{vol}_{F_0,\boldsymbol{c}_G}(U_0)$$

for $2\beta_{\text{prev}}\phi \le \frac{1}{2}$. The bound on $\mathbb{E}[\text{vol}_{F_r,\boldsymbol{c}_G}(S_1) + \cdots + \text{vol}_{F_r,\boldsymbol{c}_G}(S_r)]$ then follows from the observation that $\text{vol}_{F_r,\boldsymbol{c}_G}(S_i) = \text{vol}_{F_i,\boldsymbol{c}_G}(S_i)$ for all $i \in [r]$, since the $A_j \subseteq U_{j-1}$ which is disjoint from S_i for $j > i$. Finally, (4) follows simply from Claim A.7.22 and the linearity of expectation. □

Let us use uppercase letters (e.g., B, Λ) to denote random variables and lowercase letters (e.g., b, λ) to denote their realizations. We first derive a recurrence of $\textsc{Size}_\ell(b,\lambda)$ and $\textsc{Time}_\ell(b,\lambda)$ for the topmost layer $\ell = L$.

Lemma A.7.40. *For $\phi < O\left(\frac{\psi_0^3}{\beta_{\text{prev}}\log^3 n}\right)$ sufficiently small, we have*

$$\mathbb{E}[\textsc{Size}_L(b,\lambda)] \le 3\phi b + \max_{\mathcal{D}_1} \mathop{\mathbb{E}}_{B'\sim\mathcal{D}_1} [\mathbb{E}[\textsc{Size}_{L-1}(B',\lambda) \mid B']] + \max_{r\in\{0,\ldots,n\}} \max_{\mathcal{D}_{2,r}} \max_{\lambda_1,\ldots,\lambda_r} \mathop{\mathbb{E}}_{(B_1,\ldots,B_r)\sim\mathcal{D}_{2,r}} \left[\sum_{i\in[r]} \mathbb{E}[\textsc{Size}_L(B_i,\lambda_i) \mid B_i]\right] \tag{A.25}$$

and

$$\begin{aligned}\mathbb{E}[\textsc{Time}_L(b,\lambda)] \le \widetilde{O}\left(\frac{1}{\psi_0^2}\right)\cdot\left(\widetilde{O}\left(\frac{\lambda^2}{\phi\phi'}\right)+T_{\mathrm{prev}}(\lambda)\right)\\ +\max_{\mathcal{D}_1}\mathop{\mathbb{E}}_{B'\sim\mathcal{D}_1}[\mathbb{E}[\textsc{Time}_{L-1}(B',\lambda)\mid B']]\\ +\max_{r\in\{0,\dots,n\}}\max_{\mathcal{D}_{2,r}}\max_{\lambda_1,\dots,\lambda_r}\mathop{\mathbb{E}}_{(B_1,\dots,B_r)\sim\mathcal{D}_{2,r}}\left[\sum_{i\in[r]}\mathbb{E}[\textsc{Time}_L(B_i,\lambda_i)\mid B_i]\right],\end{aligned} \tag{A.26}$$

where $\mathcal{D}_1$ iterates over all distributions on $\{0,\dots,m\}$ with $\mathbb{E}_{B'\in\mathcal{D}_1}[B']\le b$, $\mathcal{D}_{2,r}$ iterates over all distributions on $\{0,\dots,m\}^r$ such that $\mathbb{E}_{B_1,\dots,B_r\sim\mathcal{D}_{2,r}}[B_i]\le\frac{3}{4}b$ and $\mathbb{E}_{B_1,\dots,B_r\sim\mathcal{D}_{2,r}}[B_1+\cdots+B_r]\le(1+8\phi\beta_{\mathrm{prev}})B$, and $\lambda_1,\dots,\lambda_r$ iterate over all such sequences with $\lambda_1+\cdots+\lambda_r<\lambda$.

To avoid the cumbersome expressions as in Lemma A.7.41, in the remainder of the section we will slightly overload the notation and simply write, e.g., $\mathbb{E}_{B'}$ without the preceding $\max_{\mathcal{D}_1}$, and later in the description of lemma specifies the properties that $\mathcal{D}_1$ has to satisfy. This means that we in fact consider all qualifying distributions and pick the one that maximizes the expression. As a concrete example, we will rewrite Lemma A.7.40 in the following form.

Lemma A.7.41 (Lemma A.7.40 restated)**.** *For $\phi<O\left(\frac{\psi_0^3}{\beta_{\mathrm{prev}}\log^3 n}\right)$ sufficiently small, we have*

$$\begin{aligned}\mathbb{E}[\textsc{Size}_L(b,\lambda)]\le 3\phi b+\mathop{\mathbb{E}}_{B'}[\mathbb{E}[\textsc{Size}_{L-1}(B',\lambda)\mid B']]\\ +\max_{r,\lambda_1,\dots,\lambda_r}\mathop{\mathbb{E}}_{B_1,\dots,B_r}\left[\sum_{i\in[r]}\mathbb{E}[\textsc{Size}_L(B_i,\lambda_i)\mid B_i]\right]\end{aligned} \tag{A.27}$$

and

$$\begin{aligned}\mathbb{E}[\textsc{Time}_L(b,\lambda)]\le\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot\left(\widetilde{O}\left(\frac{\lambda^2}{\phi\phi_{\mathrm{prev}}^2}\right)+T_{\mathrm{prev}}(\lambda)\right)\\ +\mathop{\mathbb{E}}_{B'}[\mathbb{E}[\textsc{Time}_{L-1}(B',\lambda)\mid B']]\\ +\max_{r,\lambda_1,\dots,\lambda_r}\mathop{\mathbb{E}}_{B_1,\dots,B_r}\left[\sum_{i\in[r]}\mathbb{E}[\textsc{Time}_L(B_i,\lambda_i)\mid B_i]\right],\end{aligned} \tag{A.28}$$

where B' is a random variable satisfying $\mathbb{E}[B']\le b$, $B_1,\dots,B_r$ are random variables satisfying $\mathbb{E}[B_i]\le\frac{3}{4}b$ and $\mathbb{E}[B_1+\cdots+B_r]\le(1+8\phi\beta_{\mathrm{prev}})b$, and $\lambda_1,\dots,\lambda_r$ iterate over all such sequences with $\lambda_1+\cdots+\lambda_r<\lambda$.

Proof. We first bound $\textsc{Size}_L(b,\lambda)$, and by the linearity of expectation it suffices to bound each of the $X^{(1)}$, $X^{(2)}$, $X^{(3)}$, and $X^{(4)}$.

- By Lemma A.7.39, $X^{(1)}$ has expected total capacities at most $3\phi b$.
- The edge set $X^{(2)}$ is empty for $\textsc{MaintainExpander}(U,L)$ as the k sampled on Line 15 is never greater than L.
- The expected total capacities of $X^{(3)}$ is $\mathbb{E}_{B'}[\mathbb{E}[\textsc{Size}_{L-1}(B',\lambda) \mid B']]$ where B' a random variable indicating the volume of the set U when calling $\textsc{MaintainExpander}(U,L-1)$ on Line 25. By Lemma A.7.39, we have $\mathbb{E}[B'] \le B$ (the volume might, with low probability, increase since the set of terminals 'F' increase).
- Finally, for $X^{(4)}$, its total capacities is equal to $\sum_{i\in[r]} \mathbb{E}[\textsc{Size}(b_i,\lambda_i,L)]$, where b_i is the volume of the cut S_i and λ_i is the number of vertices in S_i. By Lemma A.7.39, if we let B_i be the random variables for b_i, then we have $\mathbb{E}_{B_1,\dots,B_r}[B_1+\cdots+B_r] \le (1+8\phi\beta)B$. Moreover, as the S_i's are vertex-disjoint and are proper cuts, we have $\lambda_1+\cdots+\lambda_r < \lambda$. It remains to prove the expected marginal of each B_i. By (A.24) and that $2\beta\phi \le \frac{1}{2}$, we know that using the volume upper-bound of Lemma A.7.11

$$\mathbb{E}[\mathrm{vol}_{F_i,\boldsymbol{c}_G}(S_i) \mid U_{i-1},F_{i-1}] \le \frac{3}{2}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \le \frac{3}{4}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1}).$$

  By the law of total expectation and Lemma A.7.39 this implies $\mathbb{E}[\mathrm{vol}_{F_i,\boldsymbol{c}_G}(S_i)] \le \frac{3}{4}\mathrm{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ or equivalently $\mathbb{E}[B_i] \le \frac{3}{4}B$.

This proves (A.27). We likewise bound each of the four parts of $\textsc{Time}_L(b,\lambda)$ and then apply the linearity of expectation to derive (A.28). Note that among them it suffices to bound the time spent to compute $X^{(1)}$ as the other parts follow the same recurrences as their counterparts in (A.27). For this we bound the number of iterations the while-loop in $\textsc{MaintainExpander}(U,L)$ takes and then use the fact that the time spent in each iteration is dominated by calling $\textsc{CutOrEmbed}()$ (which by Lemma A.7.11 takes $O(\frac{\lambda^2}{\phi\phi'})$ time) and running $\mathcal{M}_{\mathrm{prev}}.\textsc{Cut}(D)$ (which by Claim A.7.33 takes $T_{\mathrm{prev}}(\lambda)$ time).[35] By Lemma A.7.11, we have $\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \ge \frac{1}{4t_{\mathrm{CMG}}}\cdot\frac{\psi_L}{10}\tau_U$ where $\tau_U = \frac{\psi_0^2}{64z}\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$ according to Line 7 in Algorithm A.5. This shows that $\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i) \ge \Omega\left(\frac{\psi_0^3}{\log^3 n}\right)\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$, which by (A.23) gives

$$\mathbb{E}[\mathrm{vol}_{F_i,\boldsymbol{c}_G}(U_i) \mid U_{i-1},F_{i-1}] \le \left(1-\Omega\left(\frac{\psi_0^3}{\log^3 n}\right)\right)\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(U_{i-1})$$

[35] Indeed, since a witness on U has at most $|U|^2$ edges, the time it takes to update the witnesses in $\textsc{UpdateWitness}()$ can be easily charged to the time spent in constructing/repairing them.

since $\phi < O\left(\frac{\psi_0^3}{\beta_{\text{prev}} \log^3 n}\right)$ is sufficiently small. By the law of total expectation, this shows that

$$\mathbb{E}[\text{vol}_{F_t, \boldsymbol{c}_G}(U_t)] \leq \left(1 - \Omega\left(\frac{\psi_0^3}{\log^3 n}\right)\right)^t \cdot b$$

and in particular $\mathbb{E}[\text{vol}_{F_t, |B c_G}(U_t)] \leq 1/\lambda$ for some $t = \widetilde{\Theta}\left(\frac{1}{\psi_0^3}\right)$. By Markov's inequality, this means that $\Pr[\text{vol}_{F_t, \boldsymbol{c}_G}(U_t) > 0] \leq 1/\lambda$ (note that $\text{vol}_{F_t, \boldsymbol{c}_G}(U_t)$ is a nonnegative integer). As the number of iterations is always bounded by λ because each cut removes at least one vertex from U, the expected number of iterations is $\widetilde{\Theta}\left(\frac{1}{\psi_0^3}\right) + \frac{1}{\lambda} \cdot \lambda \leq \widetilde{\Theta}\left(\frac{1}{\psi_0^3}\right)$. The expected time spent in computing $X^{(1)}$ is thus, by the linearity of expectation, bounded by $\widetilde{O}\left(\frac{1}{\psi_0^3}\right) \cdot \left(\widetilde{O}\left(\frac{\lambda^2}{\phi \phi_{\text{prev}}{}^2} + T_{\text{prev}}(\lambda)\right)\right)$. This proves (A.28). □

We can similarly write a recurrence for $\textsc{Size}_\ell(b, \lambda)$ and $\textsc{Time}_\ell(b, \lambda)$ for $\ell < L$. Recall from the beginning of this subsection that the recurrence we establish will deliberately overestimate some quantities. In other words, in case that we recurse to a bad state of the algorithm, we are still going to write the recurrence as if we are in a borderline state.

Lemma A.7.42. *For $\phi < O\left(\frac{\psi_0^3}{\beta_{\text{prev}} \log^3 n}\right)$ sufficiently small and $\ell < L$, we have*

$$\begin{aligned}
&\mathbb{E}[\textsc{Size}_\ell(b, \lambda)] \\
&\quad \leq O\left(\frac{\phi}{\psi_0^2}\right) b^{(\ell+1)/L} \\
&\qquad + \widetilde{O}\left(\frac{1}{\psi_0^4}\right) \cdot \mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L} \min\left\{1, \frac{\Delta}{b^{k/L}}\right\} \cdot \mathbb{E}[\textsc{Size}_k(b + \Delta, \lambda - 1) \mid \Delta]\right] \\
&\qquad + \mathop{\mathbb{E}}_{B'}\left[\mathbb{E}[\textsc{Size}_{\ell-1}(B', \lambda) \mid B']\right] + \max_{r, \lambda_1, \ldots, \lambda_r} \mathop{\mathbb{E}}_{B_1, \ldots, B_r}\left[\sum_{i \in [r]} \mathbb{E}[\textsc{Size}_L(B_i, \lambda_i) \mid B_i]\right]
\end{aligned} \tag{A.29}$$

and

$$\begin{aligned}
&\mathbb{E}[\textsc{Time}_\ell(b,\lambda)]\\
&\le \widetilde{O}\left(\frac{1}{\psi_0^5}\right)\cdot m^{1/L}\cdot\left(\widetilde{O}\left(\frac{\lambda^2}{\phi\phi_{\mathrm{prev}}^2\psi_0^4}\right)+T_{\mathrm{prev}}(\lambda)\right)\\
&+\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot\mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Time}_k(b+\Delta,\lambda-1)\mid\Delta]\right]\\
&+\mathop{\mathbb{E}}_{B'}\left[\mathbb{E}[\textsc{Time}_{\ell-1}(B',\lambda)\mid B']\right]+\max_{r,\lambda_1,\ldots,\lambda_r}\mathop{\mathbb{E}}_{B_1,\ldots,B_r}\left[\sum_{i\in[r]}\mathbb{E}[\textsc{Time}_L(B_i,\lambda_i)\mid B_i]\right],
\end{aligned}\tag{A.30}$$

where Δ *is a random variable with* $\mathbb{E}[\Delta]\le O\left(\frac{\phi\beta_{\mathrm{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}$, B' *is a random variable with* $\mathbb{E}[B']\le b$, $B_1,\ldots,B_r$ *satisfy* $\mathbb{E}[B_1+\cdots+B_r]\le O\left(\frac{1}{\log^{3L}n}\right)b^{(\ell+1)/L}$, *and* $\lambda_1,\ldots,\lambda_r$ *satisfy* $\sum_{i\in[r]}\lambda_i<\lambda$.

Proof. Recall that Δ_i is the units of volume added after the i-th S_i is found in the while-loop of MaintainExpander(U,ℓ). Conditioned on the total number r of the cuts found and the values of $\Delta_1,\ldots,\Delta_r$, let $\Delta\coloneqq\Delta_1+\cdots+\Delta_r$. If we call MaintainExpander(U,k) for some $k>\ell$ on Line 16, then we can upper-bound the current volume of U by $b+\Delta$. Therefore, the expected total capacities of $X^{(2)}$ is bounded by

$$\begin{aligned}
&\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta_r}{\psi_0^2\tau_U^{\ell/L}}\cdot c_{\mathrm{rb}}\ln n\right\}\cdot\mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid\Delta]\\
&\qquad\le\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{\psi_0^2\tau_U^{\ell/L}}\cdot c_{\mathrm{rb}}\ln n\right\}\cdot\mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid\Delta]
\end{aligned}\tag{A.31}$$

since the new U contains at most $\lambda-1$ vertices after removing at least one cut from it. By Definition A.7.36, the algorithm is in a borderline state when it enters the current MaintainExpander(U,ℓ) which by Definition A.7.34 means that $\frac{\psi_0^2}{128z}b\le\tau_U\le\frac{\psi_0^2}{32z}b$. Therefore, (A.31) is further upper-bounded by (if we move the ψ_0-term in the denominator out to the beginning)

$$\widetilde{O}\left(\frac{1}{\psi_0^4}\right)\cdot\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid\Delta].\tag{A.32}$$

As the bounds on $\boldsymbol{c}_G(X^{(3)})$ and $\boldsymbol{c}_G(X^{(4)})$ are the same as in Lemma A.7.41, it remains to (i) bound $\boldsymbol{c}_G(X^{(1)})$ which by lcreflemma:expected-bounds is at most $\phi\cdot\mathbb{E}[\mathrm{vol}_{F_0,\boldsymbol{c}_G}(S_1)+\cdots+\mathrm{vol}_{F_{r-1},\boldsymbol{c}_G}(S_r)]$ in expectation and (ii) use the bound to prove the expected value of Δ.

By Definition A.7.34, we know that $\|\boldsymbol{r}_{U,\ell+1}\|_1 \leq \frac{10}{\psi_{\ell+1}}\tau_U^{(\ell+1)/L}$ holds in the beginning of this current run of MaintainExpander(U,ℓ). Observe that what happened before we recurse on some MaintainExpander(X,k) is modeled by Scenario A.7.13 with $\delta_{\text{ext}} = 0$. Similar to the proof of Lemma A.7.28, conditioned on there being r cuts (i.e., none of the first $r-1$ cuts sampled a k larger than ℓ on Line 15), the probability that $\Delta_1 + \cdots + \Delta_{r-1} > \tau_U^{(\ell+1)/L}$ is bounded by

$$\prod_{i\in[r-1]}\left(1 - \min\left\{1, \frac{\Delta_i}{\psi_0^2\tau_U^{(\ell+1)/L}}\cdot c_{\text{rb}}\ln n\right\}\right) \leq \exp\left(-\sum_{i\in[r-1]}\Delta_i\cdot\frac{c_{\text{rb}}\ln n}{\psi_0^2\tau_U^{(\ell+1)/L}}\right) \leq n^{-c_{\text{rb}}}.$$

In particular, with high probability, we can apply Lemma A.7.20 on $W_{U,\ell+1}$ at the moment right after finding S_r but before we added the Δ_r units of volume. In other words, we pretend that the last cut S_r adds zero units of volume to U. Formally speaking, letting $\widetilde{\Delta}_i = \Delta_i$ for $i < r$ and $\widetilde{\Delta}_r = 0$, we can verify that Condition A.7.18 holds with $R = \frac{10}{\psi_{\ell+1}}\tau_U^{(\ell+1)/L} \leq \frac{10\psi_0}{32z}\text{vol}_{F_0,\boldsymbol{c}_G}(U_0) \leq \frac{\psi_{\ell+1}}{64}\text{vol}_{F_0,\boldsymbol{c}_G}(U_0)$ and $\widetilde{\Delta} := \widetilde{\Delta}_1 + \cdots + \widetilde{\Delta}_r = \Delta_1 + \cdots + \Delta_{r-1} \leq \tau_U^{(\ell+1)/L} \leq \frac{\psi_0^2}{32z}\text{vol}_{F_0,\boldsymbol{c}_G}(U_0) \leq \frac{\psi_{\ell+1}}{64}\text{vol}_{F_0,\boldsymbol{c}_G}(U_0)$. Thus, Lemma A.7.20 shows that $\text{vol}_{F_{r-1},\boldsymbol{c}_G}(S_1)+\cdots+\text{vol}_{F_{r-1},\boldsymbol{c}_G}(S_r) \leq \frac{80}{\psi_{\ell+1}^2}\tau_U^{(\ell+1)/L}$. As a result, conditioned on r and $\Delta_1 + \cdots + \Delta_{r-1} \leq \tau_U^{(\ell+1)/L}$, the size of $X^{(1)}$ is bounded by $\phi\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot\tau_U^{(\ell+1)/L} \leq \phi\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot b^{(\ell+1)/L}$ in expectation. As $\boldsymbol{c}_G(X^{(1)}) \leq m$ always hold, this implies via the following Fact A.7.37 that after removing the conditioning of $\Delta_1 + \cdots + \Delta_{r-1}$ we still have $\mathbb{E}[\boldsymbol{c}_G(X^{(1)}) \mid r] \leq \phi\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot b^{(\ell+1)/L} + n^{-100}$. By the law of total expectation, this implies unconditionally $\mathbb{E}[\boldsymbol{c}_G(X^{(1)})] \leq \phi\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot b^{(\ell+1)/L} + n^{-100}$.

Moreover, notice that by (A.24) we have $\mathbb{E}[\text{vol}_{F_i,\boldsymbol{c}_G}(S_i)] \leq \frac{3}{2}\text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)$ and since $\text{vol}_{F_r,\boldsymbol{c}_G}(S_i) = \text{vol}_{F_i,\boldsymbol{c}_G}(S_i) \leq \frac{3}{2}\text{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)$ we further have

$$\begin{aligned}\mathbb{E}[\text{vol}_{F_r,\boldsymbol{c}_G}(S_1) + \cdots + \text{vol}_{F_r,\boldsymbol{c}_G}(S_r)] &\leq \frac{120}{\psi_{\ell+1}^2}\tau_U^{(\ell+1)/L}\\ &\leq 120\frac{(\psi_0 b)^{(\ell+1)/L}}{\psi_{\ell+1}^2}\\ &\leq 120b^{(\ell+1)/L}\cdot\frac{\psi_0^{1/L}}{\psi_{\ell+1}^2} \leq O\left(\frac{1}{\log^{3L} n}\right)b^{(\ell+1)/L}\end{aligned}$$

by (A.20). This proves the bound on $\mathbb{E}[B_1 + \cdots + B_r]$.

It remains to bound the expected value of Δ to finish our bound on $\boldsymbol{c}_G(X^{(2)})$ via the expression (A.32) that we developed. For this we use the guarantee of $\mathcal{M}_{\text{prev}}$ and that we add at most two units of volume per edge returned by $\mathcal{M}_{\text{prev}}$.Cut$(D)$ which implies $\mathbb{E}[\Delta] \leq 2\beta_{\text{prev}}\,\mathbb{E}[\boldsymbol{c}_G(X^{(1)})] \leq O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L} + n^{-99} \leq O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}$. This proves (A.29).

For the recurrence (A.30) of $\textsc{Time}_\ell(b,\lambda)$, similar to Lemma A.7.41, it suffices to bound the number of iterations in the while-loop in $\textsc{MaintainExpander}(U,\ell)$. Again, conditioned on $\Delta_1+\cdots+\Delta_{r-1}\le\tau_U^{(\ell+1)/L}$, we have $\mathrm{vol}_{F_0,\boldsymbol{c}_G}(S_1)+\cdots+\mathrm{vol}_{F_{r-1},\boldsymbol{c}_G}(S_{r-1})\le\frac{8}{\psi_{\ell+1}^2}\tau_U^{(\ell+1)/L}$. Yet, Claim A.7.22 suggests that $\mathrm{vol}_{F_{i-1},\boldsymbol{c}_G}(S_i)\ge\frac{\psi_{\ell+1}^2\psi_\ell}{640z}\tau_U^{\ell/L}$ holds for each $i\in[r]$. Therefore, the number of iterations r can be bounded by $\widetilde{O}\left(\frac{1}{\psi_0^5}\right)\cdot m^{1/L}$, where each iteration by Lemma A.7.10 takes $\widetilde{O}\left(\frac{\lambda^2}{\phi\phi_{\mathrm{prev}}^2\psi_0^4}\right)+T_{\mathrm{prev}}(\lambda)$ time. This proves (A.30). □

Solving the Recurrences

Having established Lemmas A.7.41 and A.7.42, we can now solve the recurrences. In all of the recurrences, the only part not solvable by a simple induction is when we recurse on $\textsc{MaintainExpander}(S,L)$ (e.g., the $\max_{r,\lambda_1,\ldots,\lambda_r}\mathbb{E}_{B_1,\ldots,B_r}\left[\sum_{i\in[r]}\mathbb{E}[\textsc{Size}_L(B_i,\lambda_i)]\right]$ term in (A.27)) since the expected total volume of the cuts can grow larger than what we start with. We abstract this tricky part of the recurrence and handle it by proving the following lemma which exploits the fact that $\mathbb{E}[B_i]\le\frac{3}{4}b$.

Lemma A.7.43. *For random functions $f,g:\{0,\ldots,\mathrm{poly}(n)\}^2\to\{0,\ldots,\mathrm{poly}(n)\}$ with $f(0,\cdot)=f(\cdot,0)=g(0,\cdot)=g(\cdot,0)=0$ that admit a recurrence relationship of the form*

$$\mathbb{E}[f(b,\lambda)]\le\mathbb{E}[g(b,\lambda)]+\max_{r,\lambda_1,\ldots,\lambda_r}\mathbb{E}_{B_1,\ldots,B_r}\left[\sum_{i\in[r]}\mathbb{E}[f(B_i,\lambda_i)\mid B_i]\right]\tag{A.33}$$

where $B_1,\ldots,B_r$ are random variables satisfying $\mathbb{E}[B_i]\le\frac{3}{4}b$, $\mathbb{E}[B_1+\cdots+B_r]\le(1+\gamma)b$ for some $\gamma<O\left(\frac{1}{\log n}\right)$ sufficiently small and $\lambda_1,\ldots,\lambda_r$ satisfy $\lambda_1+\cdots+\lambda_r<\lambda$, we have

$$\mathbb{E}[f(b,\lambda)]\le\mathbb{E}[g(b,\lambda)]+\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\mathbb{E}_{\widetilde{B}_1,\ldots,\widetilde{B}_p}\left[\sum_{i\in[p]}\mathbb{E}[g(\widetilde{B}_i,\widetilde{\lambda}_i)\mid\widetilde{B}_i]\right]+n^{-100},\tag{A.34}$$

where $\widetilde{B}_1,\ldots,\widetilde{B}_p$ are random variables satisfying $\mathbb{E}[\widetilde{B}_1+\cdots+\widetilde{B}_p]\le O(\log n)b$ and $\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p$ satisfy $\widetilde{\lambda}_1+\cdots+\widetilde{\lambda}_p\le O(\log n)\lambda$ and $\widetilde{\lambda}_i<\lambda$.[36]

[36] In principle we could put the $\mathbb{E}[g(b,\lambda)]$ part into the max expression by setting $\widetilde{B}_0=b$ deterministically and $\widetilde{\lambda}_0=\lambda$. The reason why there is a standalone term is to ensure that we can later apply induction on λ for the terms in the max expression. In some future cases this may not be required (e.g., when we already have a parameter decrease that facilitates induction), we may for simplicity move the standalone term into the max expression.

Proof. Let us expand the recurrence of (A.33) and consider its recursion tree. We mark each node v in the tree with the values of b and λ, denoted by b_v and λ_v, that it corresponds to. Note that b_v is not deterministic; rather it is a realization of a random variable that we denote by B_v. Observe that it suffices to consider the part of the tree that corresponds to recursion from f to f and then in the end sum over the $g(b_v, \lambda_v)$ value for all nodes in the tree. Furthermore, we can ignore all nodes with either $b_v = 0$ or $\lambda_v = 0$ as both functions evaluate to zero on them.

By $\mathbb{E}[B_i] \le \frac{3}{4}b$ and the law of total expectation, we know that for some $d = \Theta(\log n)$ all the depth-d nodes have $\mathbb{E}[B_v] \le (3/4)^d \cdot b \le n^{-c}$ where $c > 0$ is an arbitrarily large but fixed constant. As the sum of the λ_v's decreases by at least one in each level, the number of nodes in the whole tree is bounded by $\lambda^2 \le \text{poly}(n)$ (there are at most λ levels, each consisting of at most λ nodes). Therefore, by Markov's inequality and a union bound, with high probability all depth-d nodes have $b_v = 0$, and we can safely ignore them and truncate the tree to have depth $d-1$. On the other hand, by $\mathbb{E}[B_1 + \cdots + B_r] \le (1+\gamma)b$ and again the law of total expectation, the expected value of the sum of B_v's of all depth-t nodes are bounded by $(1+\gamma)^t b$. For $\gamma < O\left(\frac{1}{\log n}\right)$ sufficiently small and $t < d$, the quantity $(1+\gamma)^t b$ is bounded by $2b$. Therefore, the expected value of the sum of B_v's in the whole recursion tree is bounded in expectation by $O(\log n)b$. Similarly, the sum of λ_v's is bounded by $O(\log n)\lambda$. This shows that conditioned on the event that the tree has depth at most $d = O(\log n)$ (which happens with high probability) we have

$$\begin{aligned}&\mathbb{E}[f(b,\lambda) \mid \text{tree has depth } d = O(\log n)]\\ &\le \mathbb{E}[g(b,\lambda)] + \max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p} \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_p} \left[\sum_{i\in[p]} \mathbb{E}[g(\widetilde{B}_i, \widetilde{\lambda}_i) \mid \widetilde{B}_i]\right],\end{aligned}$$

where each $\widetilde{B}_i$ and λ_i corresponds to the B_v and λ_v for some non-root v in the tree. The lemma now follows by Fact A.7.37 since both f and g are polynomially bounded. □

We can now simplify Lemma A.7.41 via Lemma A.7.43. Indeed, the value of $\gamma \coloneqq 8\phi\beta_{\text{prev}}$ as in Lemma A.7.41 is sufficiently smaller than $O\left(\frac{1}{\log n}\right)$ when $\phi < O\left(\frac{\psi_0^3}{\beta_{\text{prev}} \log^3 n}\right)$.

Corollary A.7.44. *For $\phi < O\left(\frac{\psi_0^3}{\beta \log^3 n}\right)$ sufficiently small, we have*

$$\begin{aligned}&\mathbb{E}[\textsc{Size}_L(b,\lambda)]\\ &\quad\le 3\phi \cdot b + \mathop{\mathbb{E}}_{B'}[\mathbb{E}[\textsc{Size}_{L-1}(B',\lambda) \mid B']]\\ &\quad+ \max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p} \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_p} \left[\sum_{i\in[p]} \mathbb{E}\left[3\phi\cdot\widetilde{B}_i + \mathop{\mathbb{E}}_{B'_i}\left[\mathbb{E}[\textsc{Size}_{L-1}(B'_i,\widetilde{\lambda}_i) \mid B'_i] \mid \widetilde{B}_i\right]\right]\right] + n^{-100}\end{aligned} \tag{A.35}$$

and

$$\begin{aligned}\mathbb{E}[\textsc{Time}_L(b,\lambda)] \le{}& \widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot\left(\widetilde{O}\left(\frac{\lambda}{\phi\phi_{\text{prev}}^2}\right)+T_{\text{prev}}(\lambda)\right)\\ &+\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot\left(\widetilde{O}\left(\frac{\widetilde{\lambda}_i^2}{\phi\phi'}\right)+T_{\text{prev}}(\widetilde{\lambda}_i)\right)\right.\right.\\ &\left.\left.+\mathop{\mathbb{E}}_{B_i'}\left[\mathbb{E}[\textsc{Time}_{L-1}(B_i',\widetilde{\lambda}_i)\mid B_i']\mid\widetilde{B}_i\right]\right]\right],\end{aligned} \tag{A.36}$$

where B' is a random variable satisfying $\mathbb{E}[B']\le b$, $\widetilde{B}_1,\ldots,\widetilde{B}_p$ are random variables satisfying $\mathbb{E}[\widetilde{B}_1+\cdots+\widetilde{B}_p]\le O(\log n)b$, $B_1',\ldots,B_p'$ are random variables satisfying $\mathbb{E}[B_i'\mid\widetilde{B}_i]\le\widetilde{B}_i$, and $\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p$ satisfy $\widetilde{\lambda}_1+\cdots+\widetilde{\lambda}_p\le O(\log n)\lambda$ and $\widetilde{\lambda}_i<\lambda$.[37]

Similarly, we can expand the term in (A.29) and (A.30) that corresponds to recursion of MaintainExpander(S,L) using Corollary A.7.44.

Corollary A.7.45. *For $\phi<O\left(\frac{\psi_0^3}{\beta_{\text{prev}}\log^3 n}\right)$ sufficiently small and $\ell<L$, we have*

$$\begin{aligned}&\mathbb{E}[\textsc{Size}_\ell(/b,\lambda)]\\ &\le O\left(\frac{\phi}{\psi_0^2}\right)b^{(\ell+1)/L}\\ &+\widetilde{O}\left(\frac{1}{\psi_0^4}\right)\mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid\Delta]\right]\\ &+\mathop{\mathbb{E}}_{B'}[\mathbb{E}[\textsc{Size}_{\ell-1}(B',\lambda)\mid B']]\\ &+\max_{q,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q}\mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_q}\left[\sum_{i\in[q]}\mathbb{E}\left[3\phi\cdot\widetilde{B}_i+\mathop{\mathbb{E}}_{B_i'}\left[\mathbb{E}[\textsc{Size}_{L-1}(B_i',\widetilde{\lambda}_i)\mid B_i']\right]\mid\widetilde{B}_i\right]\right]+n^{-100}\end{aligned} \tag{A.37}$$

[37] Note that $\textsc{Time}_L(b,\lambda)$ is always positive, and thus the n^{-100} term can be absorbed into the big-O expression in the first line. In contrast to this, $\textsc{Size}_L(b,\lambda)$ may be zero, and thus we need to explicitly put the n^{-100} term.

and

$$\begin{aligned}
\mathbb{E}[\textsc{Time}_\ell(b,\lambda)] &\le \widetilde{O}\left(\frac{1}{\psi_0^5}\right)\cdot m^{1/L}\cdot\left(\widetilde{O}\left(\frac{\lambda^2}{\phi\phi_{\text{prev}}^2\psi_0^4}\right)+T_{\text{prev}}(\lambda)\right)\\
&+\widetilde{O}\left(\frac{1}{\psi_0^4}\right)\cdot\mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Time}_k(b+\Delta,\lambda-1)\mid\Delta]\right]\\
&+\mathop{\mathbb{E}}_{B'}\left[\mathbb{E}[\textsc{Time}_{\ell-1}(B',\lambda)\mid B']\right]\\
&+\max_{q,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q}\ \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_q}\\
&\qquad\left[\sum_{i\in[q]}\mathbb{E}\left[\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot\left(\widetilde{O}\left(\frac{\widetilde{\lambda}_i^2}{\phi\phi_{\text{prev}}^2}\right)+T_{\text{prev}}(\widetilde{\lambda}_i)\right)+\mathop{\mathbb{E}}_{B_i'}\left[\mathbb{E}[\textsc{Time}_{L-1}(B_i',\widetilde{\lambda}_i)\mid B_i']\right]\mid\widetilde{B}_i\right]\right],
\end{aligned}\tag{A.38}$$

where Δ *is a random variable with* $\mathbb{E}[\Delta]\le O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}$, B' *is a random variable with* $\mathbb{E}[B']\le b$, $\widetilde{B}_1,\ldots,\widetilde{B}_q$ *are random variables satisfying* $\mathbb{E}[\widetilde{B}_1+\cdots+\widetilde{B}_q]\le O\left(\frac{1}{\log^{2L}n}\right)b^{(\ell+1)/L}$, $B_1',\ldots,B_q'$ *are random variables satisfying* $\mathbb{E}[B_i'\mid\widetilde{B}_i]\le\widetilde{B}_i$, *and* $\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q$ *satisfy* $\widetilde{\lambda}_1+\cdots+\widetilde{\lambda}_q\le O(\log n)\lambda$ *and* $\widetilde{\lambda}_i<\lambda$.

Proof. To see inequality (A.37), consider applying Corollary A.7.44 to each of the $\mathbb{E}[\textsc{Size}_L(B_i,\lambda_i)\mid B_i]$ term in (A.29). This yields

$$\begin{aligned}
&\max_{r,\lambda_1,\ldots,\lambda_r}\ \mathop{\mathbb{E}}_{B_1,\ldots,B_r}\left[\sum_{i\in[r]}\mathbb{E}[\textsc{Size}_L(B_i,\lambda_i)\mid B_i]\right]\\
&\le\mathop{\mathbb{E}}_{B_1,\ldots,B_r}\left[\sum_{i\in[r]}\mathbb{E}\left[3\phi\cdot B_i+\mathop{\mathbb{E}}_{B_i'}\left[\mathbb{E}[\textsc{Size}_{L-1}(B_i',\lambda_i)\mid B_i']\right]\right.\right.\\
&+\max_{p^{(i)},\widetilde{\lambda}_1^{(i)},\ldots,\widetilde{\lambda}_{p^{(i)}}^{(i)}}\ \mathop{\mathbb{E}}_{\widetilde{B}_1^{(i)},\ldots,\widetilde{B}_{p^{(i)}}^{(i)}}\\
&\qquad\left.\left.\left[\sum_{j\in[p^{(i)}]}\mathbb{E}\left[3\phi\cdot\widetilde{B}_j^{(i)}+\mathop{\mathbb{E}}_{B_j^{(i)\prime}}\left[\textsc{Size}_{L-1}(B_j^{(i)\prime},\widetilde{\lambda}_j^{(i)})\mid B_j^{(i)\prime}\right]\mid\widetilde{B}_j^{(i)}\right]\right]\ \middle|\ B_i\right]\right],
\end{aligned}\tag{A.39}$$

where $B_1',\ldots,B_r'$ are random variables satisfying $\mathbb{E}[B_i'\mid B_i]\le B_i$, $\widetilde{B}_1^{(i)},\ldots,\widetilde{B}_{p^{(i)}}^{(i)}$ are random variables satisfying $\mathbb{E}[\widetilde{B}_1^{(i)}+\cdots+\widetilde{B}_{p^{(i)}}^{(i)}\mid B_i]\le O(\log n)B_i$, $B_1^{(i)\prime},\ldots,B_{p^{(i)}}^{(i)\prime}$ are random variables satisfying $\mathbb{E}[B_j^{(i)\prime}\mid\widetilde{B}_j^{(i)}]\le\widetilde{B}_j^{(i)}$, and $\widetilde{\lambda}_1^{(i)},\ldots,\widetilde{\lambda}_{p^{(i)}}^{(i)}$ satisfy $\widetilde{\lambda}_1^{(i)}+\cdots+\widetilde{\lambda}_{p^{(i)}}^{(i)}\le O(\log n)\lambda_i$ and $\widetilde{\lambda}_j^{(i)}<\lambda_i$. Observe that we can combine the outer max and expectation with the inner ones, in which case if define $\widetilde{B}_0^{(i)}$ to be a random variable that always realizes to B_i and define $\widetilde{\lambda}_0^{(i)}$ to be λ_i, then we can

rewrite (A.39) as

$$\max_{q,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q} \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_q} \left[\sum_{i\in[q]} \mathbb{E}\left[3\phi\cdot\widetilde{B}_i + \mathop{\mathbb{E}}_{B'_i}\left[\mathbb{E}[\textsc{Size}_{L-1}(B'_i,\widetilde{\lambda}_i)\mid B'_i]\right] \mid \widetilde{B}_i\right]\right]$$

where q corresponds to $r+(p^{(1)}+1)+\cdots+(p^{(r)}+1)$, $\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q$ correspond to $\widetilde{\lambda}_0^{(1)},\ldots,\widetilde{\lambda}_{p^{(r)}}^{(r)}$, and $\widetilde{B}_1,\ldots,\widetilde{B}_q$ correspond to $\widetilde{B}_0^{(1)},\ldots,\widetilde{B}_{p^{(r)}}^{(r)}$. The random variables $\widetilde{B}_1,\ldots,\widetilde{B}_q$ satisfy $\mathbb{E}[\widetilde{B}_1+\cdots+\widetilde{B}_q]\le O(\log n)\,\mathbb{E}[B_1+\cdots+B_r]\le O\left(\frac{1}{\log^{2L} n}\right) b^{(\ell+1)/L}$, $B'_1,\ldots,B'_q$ satisfy $\mathbb{E}[B'_i\mid\widetilde{B}_i]\le\widetilde{B}_i$, and $\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q$ satisfy $\widetilde{\lambda}_1+\cdots+\widetilde{\lambda}_q\le O(\log n)(\lambda_1+\cdots+\lambda_r)\le O(\log n)\lambda$ and $\widetilde{\lambda}_i\le\lambda_j<\lambda$.[38] The proof of (A.38) follows analogously. □

We are finally ready to prove an actual bound on $\mathbb{E}[\textsc{Size}_\ell(b,\lambda)]$ and $\mathbb{E}[\textsc{Time}_\ell(b,\lambda)]$.

Lemma A.7.46. *For $\phi < O\left(\frac{\psi_0^{O(L^2)}}{\beta_{\text{prev}}\cdot L\cdot m^{1/L}}\right)$ sufficiently small, $\mathbb{E}[\textsc{Size}_\ell(b,\lambda)]\le O\left(\frac{1}{\psi_0^2}\right)\cdot\phi\cdot b^{(\ell+1)/L}$.*

Proof. We show that there exists a $c\ge\Theta\left(\frac{1}{\psi_0}\right)$ sufficiently large for which we have $\mathbb{E}[\textsc{Size}_\ell(b,\lambda)]\le c\phi\cdot b^{(\ell+1)/L}$. Note that it suffices to consider the case when $b\ge 1/\phi\ge O(\log n)^{O(L)}$ due to the condition on Line 2 in Algorithm A.5. We proceed by an induction on λ and ℓ. The base case of $\lambda=0$ is trivial as $\lambda=0$ implies $b=0$. Consider now $\lambda>0$ and $\ell=L$. From (A.35) and the inductive hypothesis, we have

$$\begin{aligned}
&\mathbb{E}[\textsc{Size}_L(b,\lambda)]\\
&\le 3\phi\cdot b+\mathop{\mathbb{E}}_{B'}[c\phi B']+\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[3\phi\widetilde{B}_i+\mathop{\mathbb{E}}_{B'_i}[c\phi B'_i]\mid\widetilde{B}_i\right]\right]+n^{-100}\\
&\le 3\phi\cdot b+c\phi\cdot b+\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[3\phi\widetilde{B}_i+c\phi\widetilde{B}_i\mid\widetilde{B}_i\right]\right]+n^{-100}\\
&\le (3+c)\phi\cdot b+(3+c)\phi\cdot O(\log n)b+n^{-100},
\end{aligned}$$

which is at most $c\phi\cdot b^{1+1/L}$ for $b\ge 1/\phi$ sufficiently large. To bound $\mathbb{E}[\textsc{Size}_\ell(b,\lambda)]$ for $\ell<L$, we first prove the following helper claim.

[38] We remark that the reason why we can put everything into the max and expectation is because (A.29) guarantees that $\lambda_j<\lambda$, so even though in Lemma A.7.43 we need to have a standalone term $\mathbb{E}[g(b,\lambda)]$ handling the case when there is no decrease in λ, here we can simply put λ_i (which corresponds to $\widetilde{\lambda}_0^{(i)}$) into the max and expectation expressions.

Claim A.7.47. *Assuming the inductive hypothesis, we have*

$$\mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid\Delta]\right]$$
$$\leq O(c\phi\log n)\cdot L\cdot\left(O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)\cdot b^{(\ell+2)/L}+b^{(\ell+1)/L-1/L^2}\right),$$

for $\mathbb{E}[\Delta]\leq O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}$.

Proof. By the linearity of expectation we can move the summation out of the expectation. For $k<L$, we can bound the summand as follows:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\Delta}\left[\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid\Delta]\right]\\
&\quad\leq\mathop{\mathbb{E}}_{\Delta}\left[\frac{\Delta}{b^{k/L}}\cdot\mathbb{E}[\textsc{Size}_k(2b,\lambda-1)]+\mathbb{E}[\textsc{Size}_k(2\Delta,\lambda-1)\mid\Delta]\right]\\
&\quad\leq\frac{1}{b^{k/L}}\cdot\mathbb{E}[\textsc{Size}_k(2b,\lambda-1)]\cdot\mathbb{E}[\Delta]+\underbrace{\mathop{\mathbb{E}}_{\Delta}\left[\mathbb{E}[\textsc{Size}_k(2\Delta,\lambda-1)\mid\Delta]\right]}_{(i)}\\
&\quad\leq 4c\phi\left(\frac{1}{b^{k/L}}b^{(k+1)/L}\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}+\left(O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}\right)^{(k+1)/L}\right)\\
&\quad\leq 4c\phi\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)\cdot b^{(\ell+2)/L}+4c\phi\cdot\max\left\{O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L},b^{(\ell+1)/L-1/L^2}\right\}\\
&\quad\leq 8c\phi\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)\cdot b^{(\ell+2)/L}+4c\phi\cdot b^{(\ell+1)/L-1/L^2}.
\end{aligned}$$

For $k=L$, we can no longer substitute the $\mathbb{E}[(2\Delta)^{(k+1)/L}]$ in (i) by $4\,\mathbb{E}[\Delta]^{(k+1)/L}$ since $f(x)=x^{1+\varepsilon}$ is convex for $\varepsilon>0$ and thus Jensen's inequality does not apply anymore. As such We further expand this term using Corollary A.7.44. This gives

us

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{\Delta}\left[\mathbb{E}[\textsc{Size}_L(2\Delta,\lambda-1)\mid\Delta]\right]\\
&\quad\le \mathbb{E}\left[\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\ \mathop{\mathbb{E}}_{\widetilde{\Delta}_1,\ldots,\widetilde{\Delta}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[3\phi\cdot\widetilde{\Delta}_i+\mathop{\mathbb{E}}_{\Delta'_i}\left[\mathbb{E}[\textsc{Size}_{L-1}(\Delta'_i,\widetilde{\lambda}_i)\mid\Delta'_i]\right]\,\middle|\,\widetilde{\Delta}_i\right]\right]\,\middle|\,\Delta\right]\\
&\quad\le \mathbb{E}\left[\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\ \mathop{\mathbb{E}}_{\widetilde{\Delta}_1,\ldots,\widetilde{\Delta}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[3\phi\cdot\widetilde{\Delta}_i+\mathop{\mathbb{E}}_{\Delta'_i}\left[c\phi\cdot\Delta'_i\right]\,\middle|\,\widetilde{\Delta}_i\right]\right]\,\middle|\,\Delta\right]\\
&\quad\le \mathbb{E}\left[\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\ \mathop{\mathbb{E}}_{\widetilde{\Delta}_1,\ldots,\widetilde{\Delta}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[(3+c)\phi\cdot\widetilde{\Delta}_i\right]\right]\,\middle|\,\Delta\right]\\
&\quad\le (3+c)\phi\cdot O(\log n)\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L},
\end{aligned}
$$

where $\mathbb{E}[\widetilde{\Delta}_1+\cdots+\widetilde{\Delta}_r\mid\Delta]\le O(\log n)(2\Delta)\le O(\log n)\Delta$. Substituting this back to the above calculation, we get

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{\Delta}\left[\min\left\{1,\frac{\Delta}{b}\right\}\cdot\mathbb{E}[\textsc{Size}_L(b+\Delta,\lambda-1)\mid\Delta]\right]\\
&\quad\le \frac{1}{b}\cdot\mathbb{E}[\textsc{Size}_k(2b,\lambda-1)]\cdot\mathbb{E}[\Delta]+(3+c)\phi\cdot O(\log n)\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}\\
&\quad\le O(c\phi\log n)\cdot\left(\frac{1}{b}\cdot b^{1+1/L}\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}+O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)b^{(\ell+1)/L}\right)\\
&\quad\le O(c\phi\log n)\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)\cdot b^{(\ell+2)/L}.
\end{aligned}
$$

The claim follows by summing over at most L different values of k. □

With Claim A.7.47, we can now bound $\mathbb{E}[\textsc{Size}_\ell(b,\lambda)]$ for $b\ge 1/\phi$ via Corol-

lary A.7.45 as follows:

$$\begin{aligned}
&\mathbb{E}[\textsc{Size}_\ell(b,\lambda)]\\
&\le O\left(\frac{\phi}{\psi_0^2}\right) b^{(\ell+1)/L}\\
&+ \widetilde{O}\left(\frac{1}{\psi_0^4}\right)\cdot \mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L} \min\left\{1, \frac{\Delta}{b^{k/L}}\right\}\cdot \mathbb{E}[\textsc{Size}_k(b+\Delta,\lambda-1)\mid \Delta]\right]\\
&+ \mathop{\mathbb{E}}_{B'}\left[c\phi\cdot (B')^{(\ell/L)}\right] + \max_{q,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q} \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_q}\left[\sum_{i\in[q]} \mathbb{E}\left[3\phi\cdot \widetilde{B}_i + \mathop{\mathbb{E}}_{B'_i}[c\phi\cdot B'_i]\mid \widetilde{B}_i\right]\right]\\
&\le O\left(\frac{\phi}{\psi_0^2}\right) b^{(\ell+1)/L}\\
&+ \widetilde{O}\left(\frac{1}{\psi_0^4}\right)\cdot O(c\phi\log n)\cdot L\cdot\left(O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)\cdot b^{(\ell+2)/L} + b^{(\ell+1)/L-1/L^2}\right)\\
&+ c\phi\cdot b^{\ell/L} + (3+c)\phi\cdot O\left(\frac{1}{\log^{2L} n}\right) b^{(\ell+1)/L}
\end{aligned}$$

which is at most $c\phi\cdot b^{(\ell+1)/L}$ for $c\ge\Omega\left(\frac{1}{\psi_0^2}\right)$ sufficiently large, $\phi < O\left(\frac{\psi_0^{O(L^2)}}{\beta_{\text{prev}}\cdot L\cdot m^{1/L}}\right)$ sufficiently small, and $b\ge 1/\phi\ge\widetilde{\Omega}\left(\frac{1}{\psi_0^3}\right)^{O(L^2)}$. This proves the lemma. □

The proof of $\mathbb{E}[\textsc{Time}_\ell(b,\lambda)]$ follows analogously.

Lemma A.7.48. *For $\phi < O\left(\frac{\psi_0^{O(L^2)}}{\beta_{\text{prev}}\cdot L\cdot m^{1/L}}\right)$ sufficiently small, $\mathbb{E}[\textsc{Time}_\ell(b,\lambda)]\le m^{1/L}\cdot b^{1/L}\cdot\widetilde{O}\left(\frac{\lambda^2}{\phi\phi_{\text{prev}}^2\psi_0^4} + T_{\text{prev}}(\lambda)\right)$.*

Proof. Let $G(\lambda)=\widetilde{O}\left(\frac{\lambda^2}{\phi\phi_{\text{prev}}^2\psi_0^4}+T(\lambda)\right)$ be an upper bound on the time $\textsc{MaintainExpander}(U,\ell)$ on an λ-vertex U spends in the while-loop. We proceed by an induction on λ and ℓ and show that $\mathbb{E}[\textsc{Time}_\ell(b,\lambda)]\le 2^\ell\cdot m^{1/L}\cdot b^{1/L}\cdot G(\lambda)$ for $\ell<L$ and $\mathbb{E}[\textsc{Time}_L(b,\lambda)]\le 2^L\cdot O(\log^2 n)\cdot m^{1/L}\cdot b^{1/L}\cdot G(\lambda)$. Again, due to the early return on Line 2 in Algorithm A.5, it suffices to prove the bound for $b\ge 1/\phi$ and thus we may assume $b\ge\widetilde{\Omega}\left(\frac{1}{\psi_0^4}\right)^{O(L)}$ is sufficiently large. The base case is easy to

verify. For $\ell = L$, by the inductive hypothesis, (A.36) can be bounded by

$$\begin{aligned}
&\mathbb{E}[\textsc{Time}_L(b,\lambda)] \\
&\quad\le \widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\lambda) \\
&\quad+ \max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p} \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\widetilde{\lambda}_i) + \mathop{\mathbb{E}}_{B_i'}\left[2^{L-1}\cdot m^{1/L}\cdot (B_i')^{1/L}\cdot G(\widetilde{\lambda}_i)\right]\Big|\widetilde{B}_i\right]\right] \\
&\quad\le \widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\lambda) + \widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot O(\log n)G(\lambda) + 2^{L-1}\cdot m^{1/L}\cdot O(\log n) b^{1/L}\cdot O(\log n) G(\lambda) \\
&\quad\le 2^{L-1}\cdot O(\log^2 n)\cdot m^{1/L}\cdot b^{1/L}\cdot G(\lambda)
\end{aligned}$$

since $m^{1/L} \gg \widetilde{O}\left(\frac{1}{\psi_0^3}\right)$. For $\ell < L$, we prove a helper claim similar to Claim A.7.47.

Claim A.7.49. *We have*

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\Delta}\left[\sum_{k=\ell+1}^{L}\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}\left[\textsc{Time}_k(b+\Delta,\lambda-1)\mid\Delta\right]\right] \\
&\qquad\qquad\le 2^L\cdot O(\log^2 n)\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)^{1/L}\cdot L\cdot m^{1/L}\cdot b^{1/L}\cdot G(\lambda)
\end{aligned}$$

for $\mathbb{E}[\Delta] \le O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right) b^{(\ell+1)/L}$.

Proof. We first move the summation out of the expectation and bound each summand as follows:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\Delta}\left[\min\left\{1,\frac{\Delta}{b^{k/L}}\right\}\cdot\mathbb{E}\left[\textsc{Time}_k(b+\Delta,\lambda)\mid\Delta\right]\right] \\
&\quad\le \mathop{\mathbb{E}}_{\Delta}\left[\frac{\Delta}{b^{k/L}}\cdot\mathbb{E}\left[\textsc{Time}_k(2b,\lambda)\right] + \mathbb{E}\left[\textsc{Time}_k(2\Delta,\lambda)\mid\Delta\right]\right] \\
&\quad\le 2^L\cdot O(\log^2 n)\cdot m^{1/L}\cdot\left(\frac{\mathbb{E}[\Delta]}{b^{k/L}}\cdot b^{1/L} + \mathop{\mathbb{E}}_{\Delta}[\Delta^{1/L}]\right)\cdot G(\lambda) \\
&\quad\le 2^L\cdot O(\log^2 n)\cdot m^{1/L}\cdot\left(O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right) b^{(\ell+1-k)/L}\cdot b^{1/L} + O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)^{1/L}\cdot b^{(\ell+1)/L^2}\right)\cdot G(\lambda) \\
&\quad\le 2^L\cdot O(\log^2 n)\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)^{1/L}\cdot m^{1/L}\cdot b^{1/L}\cdot G(\lambda).
\end{aligned}$$

The claim follows by summing over at most L values of k. □

We can now expand (A.38) using Claim A.7.49 and get

$$\begin{aligned}
&\mathbb{E}[\textsc{Time}_\ell(b,\lambda)] \\
&\le \widetilde{O}\left(\frac{1}{\psi_0^5}\right)\cdot m^{1/L}\cdot G(\lambda) + \widetilde{O}\left(\frac{1}{\psi_0^4}\right)\cdot 2^L\cdot O(\log^2 n)\cdot O\left(\frac{\phi\beta_{\text{prev}}}{\psi_0^2}\right)^{1/L}\cdot Lm^{1/L}b^{1/L}\cdot G(\lambda) \\
&+ \mathop{\mathbb{E}}_{B'}\left[2^{\ell-1}\cdot m^{1/L}\cdot (B')^{1/L}\cdot G(\lambda)\right] \\
&+ \underbrace{\max_{q,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q}\ \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_q}\left[\sum_{i\in[q]}\mathbb{E}\left[\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\widetilde{\lambda}_i) + \mathop{\mathbb{E}}_{B'_i}\left[2^L\cdot m^{1/L}\cdot (B'_i)^{1/L}\cdot G(\widetilde{\lambda}_i)\right]\mid \widetilde{B}_i\right]\right]}_{(i)},
\end{aligned}$$

where we can bound the last term (i) by

$$\begin{aligned}
&\widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\lambda) + 2^L\cdot m^{1/L}\cdot \max_{q,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_q}\ \mathop{\mathbb{E}}_{\widetilde{B}_1,\ldots,\widetilde{B}_q}\left[\sum_{i\in[q]}\mathbb{E}[\widetilde{B}_i]^{(1/L)}\cdot G(\widetilde{\lambda}_i)\right] \\
&\qquad\le \widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\lambda) + 2^L\cdot m^{1/L}\cdot O(\log n)\cdot G(\lambda)\cdot O\left(\frac{1}{\log^{2L} n}\right)b^{1/L} \\
&\qquad\le \widetilde{O}\left(\frac{1}{\psi_0^3}\right)\cdot G(\lambda) + \frac{m^{1/L}}{\Omega(\log n)}\cdot G(\lambda)\cdot b^{1/L}.
\end{aligned}$$

Substituting this back into the above calculation, we can see that $\mathbb{E}[\textsc{Time}_\ell(b,\lambda)]$ is at most $2^\ell\cdot m^{1/L}\cdot G(\lambda)$ for $\phi < O\left(\frac{\psi_0^{O(L^2)}}{\beta_{\text{prev}}\cdot L\cdot m^{1/L}}\right)$ sufficiently small and $b \ge 1/\phi \ge \widetilde{\Omega}\left(\frac{1}{\psi_0^5}\right)^L$. Observe that $\frac{1}{\psi_0} \gg 2^L$ by (A.20). This proves the lemma. □

To this end, we can establish the expected guarantee of $\mathcal{M}.\textsc{Init}()$ and $\mathcal{M}.\textsc{Cut}(D)$ implemented in Algorithm A.4. Recall in Definition A.7.3 that U_D is the union of U's that intersect with the input cut D.

Lemma A.7.50. *For $\frac{1}{n} < \phi < O\left(\frac{\psi_0^{O(L^2)}}{\beta_{\text{prev}}\cdot L\cdot m^{1/L}}\right)$ sufficiently small, the subroutine $\textsc{Init}(G)$ runs in expected $m^{2/L}\cdot\widetilde{O}\left(\frac{n^2}{\phi\phi_{\text{prev}}^2\psi_0^4} + T_{\text{prev}}(n)\right)$ and outputs a set X of expected size $\phi\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot m^{O(1/L)}\cdot\alpha_{\text{prev}}\cdot m$.*

Proof. We initialize F as the output of $\mathcal{M}_{\text{prev}}.\textsc{Init}(G)$ which by its guarantee satisfies $\mathbb{E}[\boldsymbol{c}_G(F)] \le \alpha_{\text{prev}}\cdot m$. Therefore, the initial volume of V on which we run $\textsc{MaintainExpander}(V,L)$ is at most $2\alpha_{\text{prev}}\cdot m$ in expectation. Observe that the algorithm is always in a good state in the beginning, and thus by Lemma A.7.38

the expected output size of $\textsc{Init}(G)$ is at most

$$\begin{aligned}\mathop{\mathbb{E}}_{F}\left[\mathbb{E}[\textsc{Size}_L(2\boldsymbol{c}_G(F),n)] \mid F\right] &\le \mathop{\mathbb{E}}_{F}\left[O\left(\frac{1}{\psi_0^2}\right)\cdot\phi\cdot(2\boldsymbol{c}_G(F))^{1+1/L}\right]\\ &\le \phi\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot m^{1/L}\cdot\alpha_{\text{prev}}\cdot m\end{aligned}$$

by Lemma A.7.46 and it runs in expected

$$\begin{aligned}\mathop{\mathbb{E}}_{F}\left[\mathbb{E}[\textsc{Time}_L(2\boldsymbol{c}_G(F),n)] \mid F)\right] &\le \mathop{\mathbb{E}}_{F}\left[m^{2/L}\cdot\widetilde{O}\left(\frac{n^2}{\phi\phi_{\text{prev}}^2\psi_0^4}+T_{\text{prev}}(n)\right)\right]\\ &\le m^{2/L}\cdot\widetilde{O}\left(\frac{n^2}{\phi\phi_{\text{prev}}^2\psi_0^4}+T_{\text{prev}}(n)\right)\end{aligned}$$

time by Lemma A.7.48. Note that $\textsc{PostProcess}(V)$ runs in $O(n^2)$ time using [Tar72] and is therefore negligible. □

Lemma A.7.51. *For* $\frac{1}{n}<\phi<O\left(\frac{\psi_0^{O(L^2)}}{\beta_{\text{prev}}\cdot L\cdot m^{1/L}}\right)$ *sufficiently small, the subroutine* $\textsc{Cut}(D)$ *in expected* $m^{2/L}\cdot\widetilde{O}\left(\frac{|U_D|^2}{\phi\phi_{\text{prev}}^2\psi_0^4}+T_{\text{prev}}(|U_D|)\right)$ *outputs a set* X *of expected total capacities* $\mathbb{E}[\boldsymbol{c}_G(X)]\le O\left(\frac{1}{\psi_0^5}\right)\cdot m^{O(1/L)}\cdot|D|$.

Proof. Recall the implementation of $\textsc{Cut}(D)$ in Algorithm A.4, where we visit each $U\in\mathcal{U}$ that intersects with D and remove the corresponding cut from U. Observe that the running time and output size can be computed for each U individually by the linearity of expectation. Fix a $U\in\mathcal{U}$. Note that U contributes to X and the running time in two ways: one is when running $\textsc{MaintainExpander}(U,k)$ on Line 13, and the other is when running $\textsc{MaintainExpander}(S,L)$ with $S=S_U$ on Line 15. We bound these two terms separately. Recall that $D_U\coloneqq D\cap G[U]$ is the set of edges that are removed from $G[U]$. Let Δ_U be *two times* the total capacities of the edge set A output by $\mathcal{M}_{\text{prev}}.\textsc{Cut}(D\cap G[U])$ which upper bounds the units of volume added to both $U\setminus S_U$ and S_U. By the guarantee of $\mathcal{M}_{\text{prev}}$, the expected value of Δ_U is upper-bounded by $2\beta_{\text{prev}}\boldsymbol{c}_G(D_U)$. Note that the k we sampled on Line 13 is from the distribution $\mathcal{R}_{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}$. Let b_U be the initial volume of U and λ_U be the number of vertices in U.

Line 13. Note that when we call $\textsc{MaintainExpander}(U,k)$ on Line 13, the volume of U is upper-bounded by $b_U+\Delta_U$. By Lemma A.7.46, conditioned on the event $\mathcal{K}$, we have $\tau_U\ge\Omega(\psi_0^2 b_U)$ and thus we can bound the expected output size of

this call by

$$\mathop{\mathbb{E}}_{\Delta_U}\left[\sum_{k=0}^{L}\min\left\{1,\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{\psi_0^2\cdot\tau_U^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b_U+\Delta_U,\lambda_U)\mid\Delta_U]\right]$$
$$\le O\left(\frac{1}{\psi_0^4}\right)\cdot\mathop{\mathbb{E}}_{\Delta_U}\left[\sum_{k=0}^{L}\min\left\{1,\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{b_U^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b_U+\Delta_U,\lambda_U)\mid\Delta_U]\right].$$

Moving the expectation into the summation, we can bound each summand by

$$\mathop{\mathbb{E}}_{\Delta_U}\left[\min\left\{1,\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{b_U^{k/L}}\right\}\cdot\mathbb{E}[\textsc{Size}_k(b_U+\Delta_U,\lambda_U)\mid\Delta_U]\right]$$
$$\le\mathop{\mathbb{E}}_{\Delta_U}\left[\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{b_U^{k/L}}\cdot\mathbb{E}[\textsc{Size}_k(2b_U,\lambda_U)]+\mathbb{E}[\textsc{Size}_k(2\Delta_U,\lambda_U)\mid\Delta_U]\right]$$
$$\le O\left(\frac{1}{\psi_0^2}\right)\cdot\phi\cdot\mathop{\mathbb{E}}_{\Delta_U}\left[\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{b_U^{k/L}}\cdot b_U^{(k+1)/L}\right]+\underbrace{\mathop{\mathbb{E}}_{\Delta_U}\left[\Delta_U^{(k+1)/L}\right]}_{(i)}$$
$$\le O\left(\frac{1}{\psi_0^2}\right)\cdot\phi\cdot\left(m^{1/L}\cdot(\boldsymbol{c}_G(D_U)/\phi+2\beta_{\text{prev}}\boldsymbol{c}_G(D_U))+\beta_{\text{prev}}\cdot\boldsymbol{c}_G(D_U)^{(k+1)/L}\right)$$
$$\le O\left(\frac{1}{\psi_0^2}\right)\cdot m^{1/L}\cdot\boldsymbol{c}_G(D_U)$$

when $k<L$. For $k=L$, as in Claim A.7.47 we likewise expand (i) using Corollary A.7.44 and get

$$\mathop{\mathbb{E}}_{\Delta_U}[\mathbb{E}[\textsc{Size}_L(2\Delta_U,\lambda_U)\mid\Delta_U]]$$
$$\le\mathbb{E}\left[\max_{p,\widetilde{\lambda}_1,\ldots,\widetilde{\lambda}_p}\mathop{\mathbb{E}}_{\widetilde{\Delta}_1,\ldots,\widetilde{\Delta}_p}\left[\sum_{i\in[p]}\mathbb{E}\left[3\phi\cdot\widetilde{\Delta}_i+\mathop{\mathbb{E}}_{\Delta_i'}\left[\mathbb{E}[\textsc{Size}_{L-1}(\Delta_i',\widetilde{\lambda}_i)\mid\Delta_i']\right]\mid\widetilde{\Delta}_i\right]\right]+n^{-100}\middle|\Delta_U\right]$$
$$\le\phi\cdot O(\log n)\cdot\mathbb{E}[\Delta_U]+O(\log n)\cdot O\left(\frac{1}{\psi_0^2}\right)\cdot\phi\cdot\mathbb{E}[\Delta_U]\le\phi\cdot O\left(\frac{\log n}{\psi_0^2}\right)\cdot\beta_{\text{prev}}\boldsymbol{c}_G(D_U).$$

Plugging this back into the above calculation we can conclude that

$$\mathop{\mathbb{E}}_{\Delta_U}\left[\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{b_U}\cdot\mathbb{E}[\textsc{Size}_L(2b_U,\lambda_U)]+\mathbb{E}[\textsc{Size}_L(2\Delta_U,\lambda_U)\mid\Delta_U]\right]$$
$$\le O\left(\frac{1}{\psi_0^2}\right)\cdot\phi\cdot\mathop{\mathbb{E}}_{\Delta_U}\left[\frac{\boldsymbol{c}_G(D_U)/\phi+\Delta_U}{b_U}\cdot B_U^{1+1/L}\right]+O\left(\frac{\phi\log n}{\psi_0^2}\right)\cdot(\beta\boldsymbol{c}_G(D_U))$$
$$\le O\left(\frac{1}{\psi_0^2}\right)\cdot m^{1/L}\cdot\boldsymbol{c}_G(D_U).$$

Summing over $O(L)$ values of k, we get that conditioned on the event $\mathcal{K}$, the expected contribution to X of Line 13 is bounded by $O\left(\frac{L}{\psi_0^6}\right)\cdot m^{1/L}\cdot\boldsymbol{c}_G(D_U)$. By Fact A.7.37

this is asymptotically the same as the unconditional expectation. The expected running time is easily bounded using Lemma A.7.48 by $m^{2/L} \cdot \widetilde{O}\left(\frac{\lambda_U^2}{\phi\phi_{\text{prev}}^2\psi_0^4} + T(\lambda_U)\right)$.

Line 15. For the contribution of Line 15, note that since F is $\frac{\phi\psi_0^2}{2}$-expanding in $(G[U], \boldsymbol{c}_G)$ by Lemma A.7.32 before the this run of $\mathcal{M}.\textsc{Cut}(D)$, the volume of S_U before adding those Δ_U units is at most $\frac{2\boldsymbol{c}_G(D_U)}{\phi\psi_0^2}$. As such, the expected output size of the $\textsc{MaintainExpander}(S_U, L)$ call on Line 15 is at most

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\Delta_U}\left[\textsc{Size}_L\left(\frac{2\boldsymbol{c}_G(D_U)}{\phi\psi_0^2} + \Delta_U, \lambda_U\right)\middle|\Delta_U\right] \\
&\leq \mathop{\mathbb{E}}_{\Delta_U}\left[\textsc{Size}_L\left(\frac{4\boldsymbol{c}_G(D_U)}{\phi\psi_0^2}, \lambda_U, L\right) + \textsc{Size}_L\left(2\Delta_U, \lambda_U\right)\middle|\Delta_U\right] \\
&\leq O\left(\frac{\phi}{\psi_0^2}\right)\left(\left(\frac{\boldsymbol{c}_G(D_U)}{\phi\psi_0^2}\right)^{1+1/L} + (\beta_{\text{prev}}\boldsymbol{c}_G(D_U))^{1+1/L}\right) \\
&\leq O\left(\frac{1}{\psi_0^6}\right) \cdot m^{O(1/L)} \cdot \boldsymbol{c}_G(D_U)
\end{aligned}$$

since $1/\phi \leq n$. The expected running time of this part, again, by Lemma A.7.48 is $m^{2/L} \cdot \widetilde{O}\left(\frac{\lambda_U^2}{\phi\phi_{\text{prev}}^2\psi_0^4} + T(\lambda_U)\right)$. Since the sum of $\boldsymbol{c}_G(D_U)$'s among all $U \in \mathcal{U}$ is at most $\boldsymbol{c}_G(D)$ and the sum of λ_U's for which $D_U \neq \emptyset$ is $|U_D|$ (recall the definition of U_D in Definition A.7.3, the lemma follows. □

This completes the discussion on expected output size and running time of the subsection.

A.7.6 Putting Everything Together

To this end, we have developed all the technical parts needed for proving Lemma A.7.4.

Lemma A.7.4. *Given a $(k, \alpha_{\text{prev}}, \beta_{\text{prev}}, \phi_{\text{prev}}, T_{\text{prev}})$-hierarchy maintainer $\mathcal{M}_{\text{prev}}$ for an n-vertex simple capacitated graph $(G, \boldsymbol{c})$, for any $L \in \mathbb{N}$ there exists some $\delta_L \leq (\log n)^{L^{O(L)}}$ such that for any $\phi < O\left(\frac{1}{\delta_L L \beta_{\text{prev}} n^{O(1/L)}}\right)$ sufficiently small we can construct a $(k+1, \alpha, \beta, \phi', T)$-hierarchy maintainer $\mathcal{M}$ with*

$$\begin{aligned}
\alpha &\leq \phi \cdot \delta_L n^{O(1/L)} \cdot \alpha_{\text{prev}}, \\
\beta &\leq \delta_L n^{O(1/L)}, \\
\phi' &\geq \min\left\{\phi_{\text{prev}}, \frac{\phi}{\delta_L}\right\}, \\
T(n) &= \delta_L n^{O(1/L)} \cdot \widetilde{O}\left(\frac{n^2}{\phi\phi_{\text{prev}}^2} + T_{\text{prev}}(n)\right).
\end{aligned} \tag{A.11}$$

Proof. The algorithm for the new hierarchy maintainer $\mathcal{M}$ is Algorithm A.4 with the given parameters L and ϕ. By Observation A.7.29, the graph $G_{\mathcal{M}}$ the algorithm maintains is equal to what its output indicates (see Definition A.7.3). Moreover, by Lemma A.7.32, with high probability, after every update F is $\frac{\phi\psi_0^2}{2}$-expanding in $(G_{\mathcal{M}}, \boldsymbol{c}_G)$. By Observation A.7.30, the terminal set F that the algorithm maintains is a superset of $G_{\mathcal{M}} \setminus G_{\mathcal{M}_{\text{prev}}}$. Letting $X \coloneqq G \setminus G_{\mathcal{M}}$, this implies if we set $\mathcal{H}_{\mathcal{M}} \coloneqq (D, X_1, \ldots, X_k, F \setminus X)$, where $\mathcal{H}_{\text{prev}} \coloneqq (D, X_1, \ldots, X_k)$ is the k-level ϕ_{prev}-expander hierarchy of $(G_{\mathcal{M}_{\text{prev}}}, \boldsymbol{c}_G)$, then it is easy to see that $\mathcal{H}_{\mathcal{M}}$ is an $\min\left\{\phi_{\text{prev}}, \frac{\phi\psi_0^2}{2}\right\}$-expander hierarchy of $(G_{\mathcal{M}}, \boldsymbol{c}_G)$ with height $k+1$. On the other hand, if with inverse polynomially small probability F is not expanding in $(G_{\mathcal{M}}, \boldsymbol{c}_G)$, then we output $X = F$ after that update and thus the $\mathcal{H}_{\mathcal{M}}$ defined above is still a valid ϕ_{prev}-expander hierarchy of $(G_{\mathcal{M}}, \boldsymbol{c}_G)$. This only affects the output size by an additive n^{-100} factor in expectation and is thus negligible. Note that we can maintain the $\mathcal{H}_{\mathcal{M}}$ after each update in time $O(|U_D|^2)$ which is subsumed by the running time of $\mathcal{M}.\textsc{Cut}(D)$ we established in Lemma A.7.51.

By Lemmas A.7.50 and A.7.51, the output edge set of $\mathcal{M}.\textsc{Init}(G)$ and $\mathcal{M}.\textsc{Cut}(D)$ has total capacities in expectation bounded by $\left(\phi \cdot O\left(\frac{1}{\psi_0^2}\right) n^{O(1/L)}\right) \cdot \alpha_{\text{prev}} \cdot m$ and $O\left(\frac{1}{\psi_0^4}\right) \cdot m^{O(1/L)} \cdot \boldsymbol{c}_G(D)$. Therefore, we have $\alpha \leq \left(\phi \cdot O\left(\frac{1}{\psi_0^2}\right) n^{O(1/L)}\right) \cdot \alpha_{\text{prev}}$ and $\beta \leq O\left(\frac{1}{\psi_0^5}\right) \cdot m^{O(1/L)}$. The subroutines run in $T(n)$ and $T(|U_D|)$ time, for $T(n) = n^{O(1/L)} \cdot \frac{1}{\psi_0^4} \cdot \widetilde{O}\left(\frac{n^2}{\phi\phi_{\text{prev}}^2} + T_{\text{prev}}(n)\right)$. Letting $\delta_L = \left(\frac{1}{\psi_0}\right)^{\Theta(L^2)} = (\log n)^{L^{\Theta(L)}}$ sufficiently large, these become the bounds stated in (A.11). □

Acknowledgements

We thank Danupon Nanongkai and Christian Wulff-Nilsen for the helpful discussions during the preliminary stages of this work.

APPENDIX

A.8 Using Dynamic Trees for Capacitated Push-Relabel

In section Section A.4, we showed Algorithm A.1. The running time analysis in Section A.4 only shows the desired running time $\tilde{O}(m + n + \sum_{e \in E} \frac{h}{\boldsymbol{w}(e)})$ (of Theorem A.4.1) when the edges are of unit capacity $\boldsymbol{c}(e) = 1$. Here we show that we can implement the same algorithm equally efficiently for any capacities, with the use of dynamic trees [ST83].

In particular, we assume the following data structure (see [ST83] for details).

Lemma A.8.1 (Dynamic Trees [ST83; GT88])**.** *There is a data structure which maintains a collection of rooted trees $\mathcal{T}$ together with values $\boldsymbol{\nu} \in \mathbb{Z}^E$ on the edges.*

The data structure supports the following operations, all in amortized $O(\log n)$ update time.

- LINK(e)*: if $e = (u, v)$, add the edge to $\mathcal{T}$, with v now the parent of u. Before this update, v must not have any parent and u cannot be in the same tree as v.*
- DELETE(e)*: remove the edge e from $\mathcal{T}$.*
- FINDMIN(u)*: find the edge e with the minimum value $\boldsymbol{\nu}(e)$ on the path from u to the root of the tree which u is in. In case of ties, return the edge closest to u.*
- ADD(u, x)*: set $\boldsymbol{\nu}(e) \leftarrow \boldsymbol{\nu}(e) + x$ for all edges e on the path from u to the root of the tree containing u.*

The idea is similar to how a standard push-relabel algorithm can be sped up with dynamic trees (see [GT88]). We keep track of a set of rooted trees $\mathcal{T}$, where for each vertex $u \in V$, we pick an arbitrary admissible out-edge $e = (u, v)$ as the parent-edge in T. Indeed this forms a rooted tree, since the parent u has lower level than v. The values $\boldsymbol{\nu}$ which the data structures keeps track of will be the residual capacities $\boldsymbol{c_f}$, from which the flow $\boldsymbol{f}$ can implicitly be calculated from.

Whenever an edge is marked admissible or inadmissible, we might need to add/remove it from the tree, and perhaps replace the removed edge with another admissible edge (this takes $O(\log n)$ time via the LINK and DELETE operations). Whenever an edge is removed from $\mathcal{T}$, we update the residual capacity of the corresponding reverse edge (which might at this point be outdated; this is okay since only one of $\overrightarrow{e}$ and $\overleftarrow{e}$ can be admissible at the same point in time, and we only need to maintain the residual capacity correctely for the admissible edge).

When no vertices can be relabeled, this means that each vertex except the unsaturated sinks will have a parent in its tree, and the roots of the trees will thus exactly be the unsaturated sinks t with $\mathbf{abs}_{\boldsymbol{f}}(t) < \boldsymbol{\nabla}(t)$. When the algorithm wants to trace a path P from s to some sink t, this path P can thus be the path from s in its tree to the corresponding root. The value of c^{augment} can be found using the FINDMIN operation. Thereafter, the residual capacities on the path P can be adjusted via the ADD operation. We still need to find all the edges on P which now has $\boldsymbol{c_f} = 0$, so that we can mark them as inadmissible. We do this with iteratively calling the FINDMIN operation climbing the path P as long as the returned edge e has $\boldsymbol{c_f} = 0$.

Except for marking the edges on P as inadmissible, we use $O(\log n)$ time per augmenting path, and by Lemma A.4.7, there are only $O(n + \sum \frac{h}{\boldsymbol{w}(e)})$ augmenting paths in total over the run of the algorithm. Lemma A.4.7 also says that each edge e appears as a saturated edge in at most $O(\frac{h}{\boldsymbol{w}(e)})$ augmenting paths, so the total cost of marking edges on augmenting paths as inadmissible, over the whole run of the algorithm, will be $O(\sum \frac{h}{\boldsymbol{w}(e)} \log n)$.

Together with the analysis in Section A.4, we conclude that we can implement Algorithm A.1 in $\tilde{O}(m+n+\sum_{e\in E}\frac{h}{\boldsymbol{w}(e)})$ time, thus proving the stated running time bound of Theorem A.4.1.

A.9 Capacity Scaling

In this section, we recall the folklore capacity scaling argument for maximum flow. In particular we say that with an additional $O(\log U)$ overhead, we can reduce capacities from $\{0,1,2,\ldots,U\}$ to $\{0,1,2,\ldots,n^2\}$.

Lemma A.9.1. *Given an algorithm $\mathcal{A}$ that can solve any maximum flow instance $\mathcal{I}=(G,\boldsymbol{c},\boldsymbol{\Delta},\boldsymbol{\nabla})$, for an n-vertex m-edge simple directed graph G, with $\|\boldsymbol{c}\|_\infty,\|\boldsymbol{\Delta}\|_\infty,\|\boldsymbol{\nabla}\|_\infty\le n^2$, in time $T_{\mathcal{A}}(n,m)$; there is an algorithm $\mathcal{A}'$ which can solve maximum flow where $\|\boldsymbol{c}\|_\infty,\|\boldsymbol{\Delta}\|_\infty,\|\boldsymbol{\nabla}\|_\infty\le U$ in time $O(T_{\mathcal{A}}(n,m)\log U)$.*

Proof. For an edge e, write $\boldsymbol{c}(e)$ in binary as $\boldsymbol{c}(e)=\sum_{i=0}^{k}2^i\cdot\boldsymbol{c}^{(i)}(e)$, similarly for a vertex v write $\boldsymbol{\Delta}(v)=\sum_{i=0}^{k}2^i\cdot\boldsymbol{\Delta}^{(i)}(v)$. and $\boldsymbol{\nabla}(v)=\sum_{i=0}^{k}2^i\cdot\boldsymbol{\nabla}^{(i)}(v)$, for $k=O(\log U)$. We will go from the most significant bit, and add one bit at a time to $\boldsymbol{c}$, $\boldsymbol{\Delta}$ and $\boldsymbol{\nabla}$. Let $\boldsymbol{c}^{(\uparrow b)}=\sum_{i=0}^{b}2^i\cdot\boldsymbol{c}^{(k-b+i)}$ be the capacity function, but we only keep the b most significant bits and scale it down. Define $\boldsymbol{\Delta}^{(\uparrow b)}$ and $\boldsymbol{\nabla}^{(\uparrow b)}$ similarly.

If $\boldsymbol{f}$ is a maximum flow of $\mathcal{I}^{(\uparrow b)}=(G,\boldsymbol{c}^{(\uparrow b)},\boldsymbol{\Delta}^{(\uparrow b)},\boldsymbol{\nabla}^{(\uparrow b)})$, we can use $\boldsymbol{f}$ as a starting point to compute the maximum flow after we added one extra bit, i.e. for the instance $\mathcal{I}^{(\uparrow b+1)}=(G,\boldsymbol{c}^{(\uparrow b+1)},\boldsymbol{\Delta}^{(\uparrow b+1)},\boldsymbol{\nabla}^{(\uparrow b+1)})$. The crucial observation is that $2\boldsymbol{f}$ is a feasible flow for $\mathcal{I}^{(\uparrow b+1)}$, and the maximum flow $\boldsymbol{f}'$ in the residual instance $\mathcal{I}_{2\boldsymbol{f}}^{(\uparrow b+1)}$ has value at most $|\boldsymbol{f}'|\le n^2$. This is since the flow instance $\mathcal{I}_{2\boldsymbol{f}}^{(\uparrow b+1)}$ is obtained from $\mathcal{I}_{\boldsymbol{f}}^{(\uparrow b)}$ (which has no more augmenting paths) by (1) doubling all the capacities, demand, and flow-values; and (2) adding up to $m\le n^2$ unit-capacity edges (and unit-source/unit-sink demand). This means that when solving $\mathcal{I}_{2\boldsymbol{f}}^{(\uparrow b+1)}$ (with algorithm $\mathcal{A}$), we may cap all capacities above by n^2, as this will not change the maximum value of the flow. We update $\boldsymbol{f}\leftarrow 2\boldsymbol{f}+\boldsymbol{f}'$, which is now a maximum flow of $\mathcal{I}^{(\uparrow b+1)}$, and proceed to the next bit. □

A.10 Omitted Proofs

Lemma A.5.3. *Given an expander hierarchy $\mathcal{H}$, in $O(m\eta)$ time we can compute an $\mathcal{H}$-respecting topological order $\boldsymbol{\tau}$.*

Proof. Let $(D,X_1,\ldots,X_\eta)=\mathcal{H}$, and recall that X_i is a separator of the graph $G_i=G\setminus X_{>i}$. By design, the collection of strongly connected components of G_i are a refinement of the strongly connected components of G_{i+1}, and $G_0=(V,D)$ is a DAG (see also Figure A.1).

To compute the $\mathcal{H}$-respecting topological order $\boldsymbol{\tau}$, we start at the highest level η and compute (in $O(m)$ time) the strongly connected components $\{C_1,C_2,\ldots,C_r\}$

of $G_\eta = G$, together with a topological order of them [Tar72]. We reorder the C_i's with respect to this topological order so that for any DAG edges $(u, v) \in D$ with $u \in C_i$ and $v \in C_j$ we have $i \le j$.

We will assign $\{1, \ldots, |C_1|\}$ to vertices in C_1, $\{|C_1|+1, \ldots, |C_1|+|C_2|\}$ to vertices in C_2 and so on, since this would guarantee that $\boldsymbol{\tau}$ is contiguous for all level-η expanders $C_1, \ldots, C_r$, and that the topological ordering $\boldsymbol{\tau}$ respects all the DAG edges between two different C_i's.

If $\eta = 0$, we are done, since each C_i's are singletons. Otherwise we may simply recurse on each strongly connected component C_i, with the hierarchy $\mathcal{H}_i = (D \cap E[C_i], X_1 \cap E[C_i], \ldots, X_{\eta-1} \cap E[C_i])$ of height $\eta(\mathcal{H}_i) = \eta - 1$, to figure out the internal ordering of the vertices inside C_i.

The total running time will be $O(m\eta)$, since on each level from η down to 0 we will need to find the strongly connected components of some graphs with a total of m edges. □

Lemma A.7.11. *Given an n-vertex strongly connected graph $G = (V, E)$, terminal edge set $F \subseteq E$, parameters ϕ, R, and a ϕ'-expander hierarchy $\mathcal{H}$ of $G \setminus F$ with height $O(\log n)$, there is an algorithm* CutOrEmbed$(G, \boldsymbol{c}_G, F, \phi, R')$ *that either output*

1. *a set $S \subseteq V$ such that $\min\{\boldsymbol{c}_G(E_G(S, \overline{S})), \boldsymbol{c}_G(E_G(\overline{S}, S))\} < \phi \cdot \mathrm{vol}_{F,\boldsymbol{c}_G}(S)$ and $\frac{1}{4t_{CMG}} R \le \mathrm{vol}_{F,\boldsymbol{c}_G}(S) \le \frac{1}{2}\mathrm{vol}_{F,\boldsymbol{c}_G}(V)$ or*
2. *a $\boldsymbol{\gamma} \in \mathbb{N}^V$ and an $(R, \phi, \widetilde{\psi})$-witness $(W, \boldsymbol{c}, \boldsymbol{r}, \Pi_{W \to G})$ of $(G, \boldsymbol{c}_G, F)$ where $\widetilde{\psi} = \Omega\left(\frac{1}{\log^3 n}\right)$ with respect to $\boldsymbol{\gamma}$.*

The algorithm runs in time $\widetilde{O}\left(\frac{n^2}{\phi\phi'^2}\right)$

Proof. We run the cut-matching game of Theorem A.3.6 with input $\boldsymbol{\nu} := \deg_{F,\boldsymbol{c}}$. Note that $\boldsymbol{\nu}(v)$ is bounded by n^2 due to capacity scaling. In each of the iteration $t_{\mathrm{CMG}} = O(\log^2 n)$ iterations, given $(\boldsymbol{\nu}_A^{(i)}, \boldsymbol{\nu}_B^{(i)})$, we invoke Theorem A.6.1 on the flow instance $\mathcal{I} = (G, \boldsymbol{c}_G, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ with $\kappa := \frac{2 \cdot c_{A.6.1}}{\phi}$ where $\boldsymbol{\Delta} := \boldsymbol{\nu}_A^{(i)}$ and $\boldsymbol{\nabla} := \boldsymbol{\nu}_B^{(i)}$. Let $\boldsymbol{f}^* := \mathbf{0}$. If the flow $\boldsymbol{f}$ from Theorem A.6.1 routes half of the demand, i.e., $|\boldsymbol{f}| \ge \frac{1}{2}\|\boldsymbol{\Delta}\|_1$, then we update $\boldsymbol{f}^* \leftarrow \boldsymbol{f}^* + \boldsymbol{f}$, $\boldsymbol{\Delta} \leftarrow \mathbf{ex}_{\boldsymbol{f}}$, and $\boldsymbol{\nabla} \leftarrow \boldsymbol{\nabla} - \mathbf{abs}_{\boldsymbol{f}}$, and re-run Theorem A.6.1 until $\|\boldsymbol{\Delta}\|_1$ becomes less than $\frac{R}{2t_{\mathrm{CMG}}}$ (which will happen in at most z runs of Theorem A.6.1, where recall that $z := 20 \log n$). On the other hand, if $|\boldsymbol{f}| < \frac{1}{2}\|\boldsymbol{\Delta}\|_1$, then $\mathbf{ex}_{\boldsymbol{f}}(V) > \frac{1}{2}\|\boldsymbol{\Delta}\|_1 \ge \frac{1}{4t_{\mathrm{CMG}}} R$. By Theorem A.6.1, in this case the cut S that it returns satisfies $\mathbf{ex}_{\boldsymbol{f}}(S) = \mathbf{ex}_{\boldsymbol{f}}(V)$ which implies $\mathrm{vol}_{F,\boldsymbol{c}_G}(S) \ge \mathbf{ex}_{\boldsymbol{f}}(S) \ge \frac{1}{4t_{\mathrm{CMG}}} R$. Likewise, we have $\mathrm{vol}_{F,\boldsymbol{c}_G}(\overline{S}) \ge \boldsymbol{\nabla}_{\boldsymbol{f}}(V) \ge \|\boldsymbol{\nabla}\|_1 - \frac{1}{2}\|\boldsymbol{\Delta}\|_1 \ge \frac{1}{2}\|\boldsymbol{\Delta}\|_1 \ge \frac{1}{4t_{\mathrm{CMG}}} R$. Therefore, the guarantee of Theorem A.6.1 implies

that

$$\begin{aligned}\boldsymbol{c}_G(E_G(S,\overline{S})) &\leq \frac{c_{A.6.1}\cdot|\boldsymbol{f}|+\min\{\mathrm{vol}_{F,\boldsymbol{c}_G}(S),\mathrm{vol}_{F,\boldsymbol{c}_G}(\overline{S})\}}{\kappa}\\ &\leq \frac{2c_{A.6.1}\cdot\min\{\mathrm{vol}_{F,\boldsymbol{c}_G}(S),\mathrm{vol}_{F,\boldsymbol{c}_G}(\overline{S})\}}{\kappa}.\end{aligned}$$

Thus, depending on whether $\mathrm{vol}_{F,\boldsymbol{c}_G}(S)\leq\mathrm{vol}_{F,\boldsymbol{c}_G}(\overline{S})$ or not we can return either S or $\overline{S}$ in Case 1.

Now, if none of the calls to Theorem A.6.1 routes less than half of the given demand, then by adding capacitated fake edges with total capacities at most $\frac{R}{2t_{\mathrm{CMG}}}$ we have found a $(\boldsymbol{\nu}_A^{(i)},\boldsymbol{\nu}_B^{(i)})$-perfect matching $(M_i,\boldsymbol{c}_i)$ in which the non-fake edges are embeddable into $(G,\boldsymbol{c}_G)$ with congestion κz. By Theorem A.3.6, after t_{CMG} iterations, we have constructed a ψ_{CMG}-expander $(\widetilde{W},\boldsymbol{c}_W)$ containing fake edges whose total capacities sum to at most $R/2$ and in which non-fake edges are embeddable into $(G,\boldsymbol{c}_G)$ with congestion $\kappa z t_{\mathrm{CMG}}$. Let $E_{\mathrm{fake}}\subseteq\widetilde{W}$ be the set of fake edges. If we set $\boldsymbol{r}\coloneqq\deg_{E_{\mathrm{fake}},\boldsymbol{c}_W}$ and $W\coloneqq\widetilde{W}\setminus E_{\mathrm{fake}}$, then we have $\|\boldsymbol{r}\|_1\leq R$, $\deg_{F,\boldsymbol{c}_G}(v)\leq\deg_{W,\boldsymbol{c}_W}(v)+\boldsymbol{r}(v)\leq t_{\mathrm{CMG}}\cdot\deg_{F,\boldsymbol{c}_G}(v)$, and that $(W,\boldsymbol{c}_W)$ embeds into $(G,\boldsymbol{c}_G)$ with congestion kzt_{CMG}. Moreover, by the expansion guarantee of $\widetilde{W}$, we have

$$\boldsymbol{c}_W(E_W(S,\overline{S}))+\boldsymbol{r}(S)\geq\boldsymbol{c}_W(E_{\widetilde{W}}(S,\overline{S}))\geq\psi_{\mathrm{CMG}}(\mathrm{vol}_{W,\boldsymbol{c}_W}(S)+\boldsymbol{r}(S))$$

and

$$\boldsymbol{c}_W(E_W(\overline{S},S))+\boldsymbol{r}(\overline{S})\geq\boldsymbol{c}_W(E_{\widetilde{W}}(\overline{S},S))\geq\psi_{\mathrm{CMG}}(\mathrm{vol}_{W,\boldsymbol{c}_W}(S)+\boldsymbol{r}(S))$$

for every $\mathrm{vol}_{W,\boldsymbol{c}_W}(S)+\boldsymbol{r}(S)\leq\mathrm{vol}_{W,\boldsymbol{c}_W}(\overline{S})+\boldsymbol{r}(\overline{S})$. As such, $(W,\boldsymbol{c}_W,\boldsymbol{r},\Pi_{W\to G})$ is an $(R,\phi,\widetilde{\psi})$-witness of $(G,\boldsymbol{c}_G,F)$ with respect to $\boldsymbol{\gamma}(S)\coloneqq\mathrm{vol}_{W,\boldsymbol{c}_W}(S)+\boldsymbol{r}(S)$ for some $\widetilde{\psi}=\Omega\left(\frac{1}{\log^2 n}\right)$. The running time of the algorithm is $\widetilde{O}\left(\frac{n^2\kappa}{\phi'^2}\right)$ for $\kappa=\frac{2\cdot c_{A.6.1}}{\phi}$ which is $\widetilde{O}\left(\frac{n^2}{\phi\phi'^2}\right)$. This proves the lemma. □

Lemma A.7.19. *Given a graph $G=(V,E)$ and a sequence of cuts $S_1,\ldots,S_k$ where $S_i\subseteq V_{i-1}$ with $V_i\coloneqq V_{i-1}\setminus S_i$ and $V_0\coloneqq V$ satisfies*

$$\sum_{i\in[k]}\min\left\{\boldsymbol{c}_G(E_{G[V_{i-1}]}(S_i,\overline{S_i})),\boldsymbol{c}_G(E_{G[V_{i-1}]}(\overline{S_i},S_i))\right\}<\phi\cdot\sum_{i\in[k]}\mathrm{vol}_{F,\boldsymbol{c}_G}(S_i)\quad(A.18)$$

and

$$\sum_{i\in[k]}\mathrm{vol}_{F,\boldsymbol{c}_G}(S_i)\leq\alpha\cdot\mathrm{vol}_{F,\boldsymbol{c}_G}(V),$$

there is a $\left(\min\left\{\frac{\alpha}{2},1-\alpha\right\}\mathrm{vol}_{F,\boldsymbol{c}_G}(V)\right)$-balanced $\left(2\phi\min\left\{1,\frac{\alpha}{1-\alpha}\right\}\right)$-sparse cut in $(G,\boldsymbol{c})$ with respect to F.

Proof of Lemma A.7.19. Let us call an S_i out-sparse if $\boldsymbol{c}_G\left(E_{G[V_{i-1}]}(S_i, \overline{S_i})\right) \leq \boldsymbol{c}_G\left(E_{G[V_{i-1}]}(\overline{S_i}, S_i)\right)$ and in-sparse otherwise. Let $\mathcal{I}_{\text{out}} \coloneqq \{i : S_i \text{ is out-sparse}\}$ and $\mathcal{I}_{\text{in}} \coloneqq \{i : S_i \text{ is in-sparse}\}$. Let $S_{\text{out}} \coloneqq \bigcup_{i\in\mathcal{I}_{\text{out}}} S_i$ and $S_{\text{in}} \coloneqq \bigcup_{i\in\mathcal{I}_{\text{in}}} S_i$. Suppose without loss of generality that $\text{vol}_{F,\boldsymbol{c}_G}(S_{\text{out}}) \geq \text{vol}_{F,\boldsymbol{c}_G}(S_{\text{in}})$. By Observation A.7.15 we have

$$E_G(S_{\text{out}}, \overline{S_{\text{out}}}) \subseteq \bigcup_{i\in\mathcal{I}_{\text{out}}} E_{G[V_{i-1}]}(S_i, \overline{S_i}) \cup \bigcup_{i\in\mathcal{I}_{\text{in}}} E_{G[V_{i-1}]}(\overline{S_i}, S_i)$$

and therefore $\boldsymbol{c}_G\left(E_G(S_{\text{out}}, \overline{S_{\text{out}}})\right) < 2\phi \cdot \text{vol}_{F,\boldsymbol{c}_G}(S_{\text{out}})$. Since $\text{vol}_{F,\boldsymbol{c}_G}(S_{\text{out}}) \geq \frac{\alpha}{2}\text{vol}_{F,\boldsymbol{c}_G}(V)$ and $\text{vol}_{F,\boldsymbol{c}_G}(\overline{S_{\text{out}}}) \geq (1-\alpha)\text{vol}_{F,\boldsymbol{c}_G}(V)$, the lemma follows. □

Lemma A.7.35. *Conditioned on the algorithm being in a good state at the current moment, with high probability, the algorithm will remain in borderline states until termination.*

Proof. Recall that $\mathcal{K}$ is the event that $\Delta^{(U,\ell)}(t_1,t_2) \leq \frac{\psi_\ell}{10}\tau_U^{\ell/L}$ and $\delta_{\text{ext}}^{(U,\ell)}(t_1,t_2) \leq \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/L}$ hold for all active tuples (U,ℓ,t_1,t_2) which happens with high probability by Lemma A.7.27. Let t_{start} be the moment in the lemma statement when the algorithm is in a good state. Note that the random choices that happened after time t_{start} are completely independent of the condition that the algorithm is in a good state at time t_{start}. Therefore, Lemma A.7.27 suggests that with high probability $\Delta^{(U,\ell)}(t_1,t_2) \leq \frac{\psi_\ell}{10}\tau_U^{\ell/L}$ and $\delta_{\text{ext}}^{(U,\ell)}(t_1,t_2) \leq \frac{\phi\psi_\ell^2}{40}\tau_U^{\ell/L}$ hold for all active tuples (U,ℓ,t_1,t_2) with $t_1 \geq t_{\text{start}}$.

Similar to Lemma A.7.28, we prove by induction on time starting from t_{start}. Let $t \geq t_{\text{start}}$ be the current time. Fix a $U \in \mathcal{U}$ and $\ell \in \{0,\dots,L\}$ for which $W_{U,\ell}$ is not currently being rebuilt. Consider the last time $t_{\text{last}}^{(U,\ell)}$ that it was rebuilt, and let $\widetilde{t_{\text{last}}^{(U,\ell)}} \coloneqq \max\{t_{\text{last}}^{(U,\ell)}, t_{\text{start}}\}$. Let U_0 and F_0 be the set U and F at time $\widetilde{t_{\text{last}}^{(U,\ell)}}$. Note that if $t_{\text{last}}^{(U,\ell)} \geq t_{\text{start}}$, then by Lemma A.7.21 the witness of U and ℓ at time $\widetilde{t_{\text{last}}^{(U,\ell)}}$ satisfies $\|\boldsymbol{r}_{U_0,\ell}\|_1 \leq \frac{\psi_\ell}{10}\tau_U^{\ell/L}$. On the other hand, if $t_{\text{last}}^{(U,\ell)} < t_{\text{start}}$, then by the lemma statement that the algorithm is in a good state (where recall the definition of good in Definition A.7.34), we have $\|\boldsymbol{r}_{U_0,\ell}\|_1 \leq \tau_U^{\ell/L}$ (observe that $W_{U,\ell}$ *cannot* be being rebuilt at time t_{start} in this case, as that would imply by Observation A.7.24 that either it is still currently being rebuilt or $t_{\text{last}}^{(U,\ell)} \geq t_{\text{start}}$). In either case, we have $\|\boldsymbol{r}_{U_0,\ell}\|_1 \leq \tau_U^{\ell/L}$.

To likewise apply Lemma A.7.20, we note again that what happened from $\widetilde{t_{\text{last}}^{(U,\ell)}}$ to the current moment is modeled by Scenario A.7.13. Moreover, Condition A.7.18(i) and (ii) hold by exactly the same arguments as in the proof of Lemma A.7.28. It thus remains to verify Condition A.7.18(iii). Again, $\delta_\ell \leq \frac{1}{16}$ is straightforward. That $R \leq \frac{\psi_\ell}{64}\text{vol}_{F_0}(U_0)$ is by $R \leq \tau_U^{\ell/L} \leq \frac{\psi_0^2}{32z}\text{vol}_{F_0}(U_0) \leq \frac{\psi_\ell}{64}\text{vol}_{F_0}(U_0)$ by the

inductive hypothesis at time $\widetilde{t^{(U,L)}_{\text{last}}}$. The bound on $\Delta \coloneqq \Delta^{(U,\ell)}(\widetilde{t^{(U,\ell)}_{\text{last}}}, t)$ and $\delta_{\text{ext}} \coloneqq \delta^{(U,\ell)}_{\text{ext}}(\widetilde{t^{(U,\ell)}_{\text{last}}}, t)$ follow from our discussion in the beginning of this proof.

Since Condition A.7.18 holds, Lemma A.7.20 applied on $W_{U,\ell}$ (with $R \coloneqq \tau_U^{\ell/L}$, $\Delta \coloneqq \Delta^{(U,\ell)}(\widetilde{t^{(U,\ell)}_{\text{last}}}, t)$, and $\delta_{\text{ext}} \coloneqq \delta^{(U,\ell)}_{\text{ext}}(\widetilde{t^{(U,\ell)}_{\text{last}}}, t)$) shows that $\|\boldsymbol{r}_{U,\ell}\|_1 \leq \frac{4(R+\Delta)}{\psi_\ell} + \frac{8}{\psi_\ell^2 \phi}\delta_{\text{ext}} \leq \frac{10}{\psi_\ell}\tau_U^{\ell/L}$.

For the bound on τ_U, we consider when $\ell = L$. A similar argument as in the proof of Lemma A.7.28 shows that the current volume of U is at least $\frac{64z}{\psi_0}\tau_U - \frac{10}{\psi_0^2}\tau_U \geq \frac{32z}{\psi_0^2}\tau_U$. Likewise, the volume increase by at most $\Delta^{(U,L)}(\widetilde{t^{(U,L)}_{\text{last}}}, t)$, hence the lower bound of $\tau_U \geq \frac{\psi_0^2}{128z}\text{vol}_F(U)$. □

Bibliography

[ALPS23] Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. "All-Pairs Max-Flow is no Harder than Single-Pair Max-Flow: Gomory-Hu Trees in Almost-Linear Time". In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. IEEE, 2023, pp. 2204–2212. DOI: 10.1109/FOCS57990.2023.00137 (cit. on p. 71).

[BBPNSSS22] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. "Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 20:1–20:20. DOI: 10.4230/LIPICS.ICALP.2022.20 (cit. on p. 71).

[BCPKLPSS23] Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. "A Deterministic Almost-Linear Time Algorithm for Minimum-Cost Flow". In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. IEEE, 2023, pp. 503–514. DOI: 10.1109/FOCS57990.2023.00037 (cit. on p. 72).

[BEHKKPSS10] Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondrej Pangrác, Heiko Schilling, and Martin Skutella. "Length-bounded cuts and flows". In: *ACM Trans. Algorithms* 7.1 (2010), 4:1–4:27. DOI: 10.1145/1868237.1868241 (cit. on p. 94).

[Ber17] Aaron Bernstein. "Deterministic Partially Dynamic Single Source Shortest Paths in Weighted Graphs". In: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 44:1–44:14. DOI: 10.4230/LIPICS.ICALP.2017.44 (cit. on p. 77).

[BGJLLPS22] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. "Faster maxflow via improved dynamic spectral vertex sparsifiers". In: *STOC*. ACM, 2022, pp. 543–556. DOI: 10.1145/3519935.3520068 (cit. on p. 72).

[BLLSSSW21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*. ACM, 2021, pp. 859–869. DOI: 10.1145/3406325.3451108 (cit. on p. 71).

[BLNPSSSW20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 919–930. DOI: 10.1109/FOCS46700.2020.00090 (cit. on p. 71).

[BPS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. "Deterministic Decremental Reachability, SCC, and Shortest Paths via Directed Expanders and Congestion Balancing". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 1123–1134. DOI: 10.1109/FOCS46700.2020.00108 (cit. on pp. 73, 75, 79, 85, 87, 89, 129–132, 134, 135).

[CGHPS20] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. "Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 1135–1146. DOI: 10.1109/FOCS46700.2020.00109 (cit. on p. 71).

[CHLP23] Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. "Steiner Connectivity Augmentation and Splitting-off in Poly-logarithmic Maximum Flows". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*. SIAM, 2023, pp. 2449–2488. DOI: 10.1137/1.9781611977554.CH95 (cit. on p. 71).

[CK24a] Julia Chuzhoy and Sanjeev Khanna. "A Faster Combinatorial Algorithm for Maximum Bipartite Matching". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*. SIAM, 2024, pp. 2185–2235 (cit. on p. 72).

[CK24b] Julia Chuzhoy and Sanjeev Khanna. "Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm". In: *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2024*. ACM, 2024 (cit. on p. 72).

[CKLPGS22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *FOCS*. IEEE, 2022, pp. 612–623. DOI: 10.1109/FOCS54457.2022.00064 (cit. on p. 72).

[CKMST11] Paul F. Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. "Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs". In: *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*. ACM, 2011, pp. 273–282. DOI: 10.1145/1993636.1993674 (cit. on p. 71).

[CLNPSQ21] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. "Minimum Cuts in Directed Graphs via Partial Sparsification". In: *FOCS*. IEEE, 2021, pp. 1147–1158. DOI: 10.1109/FOCS52979.2021.00113 (cit. on p. 71).

[Dan51] George B Dantzig. "Application of the simplex method to a transportation problem". In: *Activity analysis and production and allocation* (1951) (cit. on p. 71).

[DGGP19] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. "Fully dynamic spectral vertex sparsifiers and applications". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*. ACM, 2019, pp. 914–925. DOI: 10.1145/3313276.3316379 (cit. on p. 71).

[Dij59] Edsger W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271. DOI: 10.1007/BF01386390 (cit. on p. 113).

[Din70] Efim A Dinic. "Algorithm for solution of a problem of maximum flow in networks with power estimation". In: *Soviet Math. Doklady*. Vol. 11. 1970, pp. 1277–1280 (cit. on p. 71).

[DS08] Samuel I. Daitch and Daniel A. Spielman. "Faster approximate lossy generalized flow via interior point algorithms". In: *STOC*. ACM, 2008, pp. 451–460 (cit. on p. 71).

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (1972), pp. 248–264. DOI: 10.1145/321694.321699 (cit. on p. 71).

[ET75] Shimon Even and Robert Endre Tarjan. "Network Flow and Testing Graph Connectivity". In: *SIAM J. Comput.* 4.4 (1975), pp. 507–518. DOI: 10.1137/0204043 (cit. on pp. 71, 72).

[FF56] Lester R. Ford and Delbert R. Fulkerson. "Maximal flow through a network". In: *Canadian journal of Mathematics* 8.3 (1956), pp. 399–404 (cit. on p. 71).

[Fin18] Jeremy T. Fineman. "Nearly work-efficient parallel algorithm for digraph reachability". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*. ACM, 2018, pp. 457–470 (cit. on p. 75).

[GH61] Ralph E Gomory and Tien Chung Hu. "Multi-terminal network flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 551–570 (cit. on p. 71).

[GKRSY03] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, F. Bruce Shepherd, and Mihalis Yannakakis. "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems". In: *J. Comput. Syst. Sci.* 67.3 (2003), pp. 473–496. DOI: 10.1016/S0022-0000(03)00066-7 (cit. on p. 94).

[GLP21] Yu Gao, Yang P. Liu, and Richard Peng. "Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao". In: *FOCS*. IEEE, 2021, pp. 516–527. DOI: 10.1109/FOCS52979.2021.00058 (cit. on p. 72).

[GR98] Andrew V. Goldberg and Satish Rao. "Beyond the Flow Decomposition Barrier". In: *J. ACM* 45.5 (1998), pp. 783–797. DOI: 10.1145/290179.290181 (cit. on pp. 71, 72).

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*. SIAM, 2021, pp. 2212–2228. DOI: 10.1137/1.9781611976465.132 (cit. on pp. 73, 74, 79, 80, 84, 101).

[GT88] Andrew V. Goldberg and Robert Endre Tarjan. "A new approach to the maximum-flow problem". In: *J. ACM* 35.4 (1988), pp. 921–940. DOI: 10.1145/48014.61051 (cit. on pp. 71, 72, 88, 93, 178, 179).

[HKPW23] Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. "Maintaining Expander Decompositions via Sparse Cuts". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*. SIAM, 2023, pp. 48–69. DOI: 10.1137/1.9781611977554.CH2 (cit. on pp. 73, 85, 87, 89, 129–131, 134–137, 139, 140, 142, 148, 152).

[HRW17] Monika Henzinger, Satish Rao, and Di Wang. "Local Flow Partitioning for Faster Edge Connectivity". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*. SIAM, 2017, pp. 1919–1938. DOI: 10.1137/1.9781611974782.125 (cit. on pp. 85, 113).

[Kar73] Alexander V Karzanov. "On finding maximum flows in networks with special structure and some applications". In: *Matematicheskie Voprosy Upravleniya Proizvodstvom* 5 (1973), pp. 81–94 (cit. on pp. 71, 72).

[KL15] David R. Karger and Matthew S. Levine. "Fast Augmenting Paths by Random Sampling from Residual Graphs". In: *SIAM J. Comput.* 44.2 (2015), pp. 320–339. DOI: 10.1137/070705994 (cit. on p. 71).

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*. SIAM, 2014, pp. 217–226. DOI: 10.1137/1.9781611973402.16 (cit. on pp. 71, 75).

[KLS20] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. "Unit Capacity Maxflow in Almost $O(m^{4/3})$ Time". In: *FOCS*. IEEE, 2020, pp. 119–130. DOI: 10.1109/FOCS46700.2020.00020 (cit. on p. 71).

[KMP12] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. "Faster approximate multicommodity flow using quadratically coupled flows". In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*. ACM, 2012, pp. 1–18. DOI: 10.1145/2213977.2213979 (cit. on p. 71).

[KP22] Shimon Kogan and Merav Parter. "New Diameter-Reducing Shortcuts and Directed Hopsets: Breaking the Barrier". In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*. SIAM, 2022, pp. 1326–1341. DOI: 10.1137/1.9781611977073.55 (cit. on p. 75).

[KRV06] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. "Graph partitioning using single commodity flows". In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC 2006*. ACM, 2006, pp. 385–390. DOI: 10.1145/1132516.1132574 (cit. on pp. 85, 89, 92, 93).

[LJS19] Yang P. Liu, Arun Jambulapati, and Aaron Sidford. "Parallel Reachability in Almost Linear Work and Square Root Depth". In: *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*. IEEE Computer Society, 2019, pp. 1664–1686 (cit. on p. 75).

[LNPSY21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. "Vertex connectivity in poly-logarithmic max-flows". In: *STOC*. ACM, 2021, pp. 317–329. DOI: 10.1145/3406325.3451088 (cit. on p. 71).

[Lou10] Anand Louis. "Cut-Matching Games on Directed Graphs". In: *CoRR* abs/1010.1047 (2010). arXiv: 1010.1047 (cit. on pp. 74, 85, 89, 92, 93, 132).

[LP20] Jason Li and Debmalya Panigrahi. "Deterministic Min-cut in Poly-logarithmic Max-flows". In: *FOCS*. IEEE, 2020, pp. 85–92. DOI: 10.1109/FOCS46700.2020.00017 (cit. on p. 71).

[LS14] Yin Tat Lee and Aaron Sidford. "Path Finding Methods for Linear Programming: Solving Linear Programs in Õ(sqrt(rank)) Iterations and Faster Algorithms for Maximum Flow". In: *FOCS*. 2014, pp. 424–433. DOI: 10.1109/FOCS.2014.52 (cit. on p. 71).

[LS20] Yang P. Liu and Aaron Sidford. "Faster energy maximization for faster maximum flow". In: *STOC*. ACM, 2020, pp. 803–814 (cit. on p. 71).

[Mad13] Aleksander Madry. "Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back". In: *FOCS*. IEEE Computer Society, 2013, pp. 253–262. DOI: 10.1109/FOCS.2013.35 (cit. on p. 71).

[Mad16] Aleksander Madry. "Computing maximum flow with augmenting electrical flows". In: *FOCS*. IEEE. 2016, pp. 593–602. DOI: 10.1109/FOCS.2016.70 (cit. on p. 71).

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. "Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2 - \epsilon})$-time". In: *STOC*. ACM, 2017, pp. 1122–1129. DOI: 10.1145/3055399.3055447 (cit. on pp. 85, 131, 135).

[NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. "Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time". In: *FOCS*. IEEE Computer Society, 2017, pp. 950–961. DOI: 10.1109/FOCS.2017.92 (cit. on pp. 85, 129, 134, 135).

[Pen16] Richard Peng. "Approximate Undirected Maximum Flows in $O(m\text{polylog}(n))$ Time". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*. SIAM, 2016, pp. 1862–1867. DOI: 10.1137/1.9781611974331.ch130 (cit. on pp. 71, 75).

[PT07] Mihai Puatracscu and Mikkel Thorup. "Planning for Fast Connectivity Updates". In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*. IEEE Computer Society, 2007, pp. 263–271. DOI: 10.1109/FOCS.2007.54 (cit. on pp. 73, 79, 80, 84, 101).

[PW20] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. "Decremental SSSP in Weighted Digraphs: Faster and Against an Adaptive Adversary". In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*. SIAM, 2020, pp. 2542–2561. DOI: 10.1137/1.9781611975994.155 (cit. on p. 77).

[Räc02] Harald Räcke. "Minimizing Congestion in General Networks". In: *43rd Symposium on Foundations of Computer Science (FOCS 2002)*. IEEE Computer Society, 2002, pp. 43–52. DOI: 10.1109/SFCS.2002.1181881 (cit. on pp. 73, 80).

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. "Computing Cut-Based Hierarchical Decompositions in Almost Linear Time". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*. SIAM, 2014, pp. 227–238. DOI: 10.1137/1.9781611973402.17 (cit. on pp. 73, 74, 84).

[She13] Jonah Sherman. "Nearly Maximum Flows in Nearly Linear Time". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*. IEEE Computer Society, 2013, pp. 263–269. DOI: 10.1109/FOCS.2013.36 (cit. on pp. 71, 75).

[She17] Jonah Sherman. "Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*. ACM, 2017, pp. 452–460. DOI: 10.1145/3055399.3055501 (cit. on pp. 71, 75).

[SP24] Aurelio L. Sulser and Maximilian Probst Gutenberg. "A Simple and Near-Optimal Algorithm for Directed Expander Decompositions". In: *CoRR* abs/2403.04542 (2024). DOI: 10.48550/ARXIV.2403.04542. arXiv: 2403.04542 (cit. on pp. 73, 89, 130, 134).

[ST04] Daniel A. Spielman and Shang-Hua Teng. "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. ACM, 2004, pp. 81–90. DOI: 10.1145/1007352.1007372 (cit. on p. 71).

[ST18] Aaron Sidford and Kevin Tian. "Coordinate Methods for Accelerating ℓ_∞ Regression and Faster Approximate Maximum Flow". In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*. IEEE Computer Society, 2018, pp. 922–933. DOI: 10.1109/FOCS.2018.00091 (cit. on pp. 71, 75).

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. "A Data Structure for Dynamic Trees". In: *J. Comput. Syst. Sci.* 26.3 (1983), pp. 362–391. DOI: 10.1016/0022-0000(83)90006-5 (cit. on pp. 73, 74, 88, 99, 178).

[SW19] Thatchaphol Saranurak and Di Wang. "Expander Decomposition and Pruning: Faster, Stronger, and Simpler". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*. SIAM, 2019, pp. 2616–2635. DOI: 10.1137/1.9781611975482.162 (cit. on pp. 74, 85, 87, 113, 130, 134).

[Tar72] Robert Endre Tarjan. "Depth-First Search and Linear Graph Algorithms". In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010 (cit. on pp. 102, 175, 181).

[Wul17] Christian Wulff-Nilsen. "Fully-dynamic minimum spanning forest with improved worst-case update time". In: *STOC*. ACM, 2017, pp. 1130–1143. DOI: 10.1145/3055399.3055415 (cit. on pp. 85, 131, 135).

Paper B

Breaking the Quadratic Barrier for Matroid Intersection

Joakim Blikstad, Jan van den Brand,
Sagnik Mukhopadhyay, Danupon Nanongkai

Article published in STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021. [BBMN21]
Full version at `https://arxiv.org/abs/2102.05548`.

Abstract

The matroid intersection problem is a fundamental problem that has been extensively studied for half a century. In the classic version of this problem, we are given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on a comment ground set V of n elements, and then we have to find the largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ by making *independence oracle queries* of the form "Is $S \in \mathcal{I}_1$?" or "Is $S \in \mathcal{I}_2$?" for $S \subseteq V$. The goal is to minimize the number of queries.

Beating the existing $\tilde{O}(n^2)$ bound, known as the *quadratic barrier*, is an open problem that captures the limits of techniques from two lines of work. The first one is the classic Cunningham's algorithm [SICOMP 1986], whose $\tilde{O}(n^2)$-query implementations were shown by CLS+ [FOCS 2019] and Nguy˜ên [2019].[1] The other one is the general cutting plane method of Lee, Sidford, and Wong [FOCS 2015]. The only progress towards breaking the quadratic barrier requires either *approximation* algorithms or a more powerful *rank oracle query* [CLS+ FOCS 2019]. No exact algorithm with $o(n^2)$ independence queries was known.

In this work, we break the quadratic barrier with a randomized algorithm guaranteeing $\tilde{O}(n^{9/5})$ independence queries with high probability, and a deterministic algorithm guaranteeing $\tilde{O}(n^{11/6})$ independence queries. Our key insight is simple and fast algorithms to solve a graph reachability problem that arose in the standard augmenting path framework [Edmonds 1968]. Combining this with previous exact and approximation algorithms leads to our results.

[1] More generally, these algorithms take $\tilde{O}(nr)$ queries where r denotes the rank which can be as big as n.

B.1 Introduction

Matroid intersection. The matroid intersection problem is a fundamental combinatorial optimization problem that has been studied for over half a century. A wide variety of prominent optimization problems, such as bipartite matching, finding an arborescence, finding a rainbow spanning tree, and spanning tree packing, can be modeled as matroid intersection problems [Sch03, Chapter 41]. Hence the matroid intersection problem is a natural avenue to study all of these problems simultaneously.

Formally, a matroid is defined by the the tuple $\mathcal{M} = (V, \mathcal{I})$ where V is a finite set of size n, called the *ground set*, and $\mathcal{I} \subseteq 2^V$ is a family of subsets of V, known as the *independent sets*, that satisfy two properties: (i) $\mathcal{I}$ is *downward closed*, i.e., all subsets of any set in $\mathcal{I}$ are also in $\mathcal{I}$, and (ii) for any two sets $A, B \in \mathcal{I}$ with $|A| < |B|$, there is an element $v \in B \setminus A$ such that $A \cup \{v\} \in \mathcal{I}$, i.e., A can be *extended* by an element in B. Given two such matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined over the same ground set V, the matroid intersection problem asks to output the largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. The size of such a set is called *rank* and is denoted by r.

The classic version of this problem that has been studied since the 1960s assumes *independence query* access to the matroids: Given a matroid $\mathcal{M}$, an independence oracle takes a set $S \subseteq V$ as input and outputs a single boolean bit depending on whether $S \in \mathcal{I}$ or not, i.e., it outputs 1 iff $S \in \mathcal{I}$. The matroid intersection problem assumes the existence of two such independence oracles, one for each matroid. The goal is to design an efficient algorithm in order to minimize the number of such oracle accesses, i.e., to minimize the independence query complexity of the matroid intersection problem. This is the version of the problem that we study in this work. Note that a more powerful query model called *rank query* has been recently studied in [LSW15; CLSSW19]. We do *not* consider such model.

Previous work. Starting with the work of Edmonds in the 1960s, algorithms with polynomial query complexity for matroid intersection have been studied [EDVJ68; Edm70; AD71; Law75; Edm79; Cun86; LSW15; Ngu19; CLSSW19]. In 1986, Cunningham [Cun86] designed an algorithm with query complexity $O(nr^{1.5})$ based on the "blocking flow" ideas similar to Hopcroft-Karp's bipartite-matching algorithm or Dinic's maximum flow algorithm. This was the best query algorithm for the matroid intersection problem for close to three decades until the recent works of Nguyen [Ngu19] and Chakrabarty-Lee-Sidford-Singla-Wong [CLSSW19] who independently showed that Cunningham's algorithm can be implemented using only $\tilde{O}(nr)$ independence queries. In a separate line of work, Lee-Sidford-Wong [LSW15] proposed a cutting plane algorithm using $\tilde{O}(n^2)$ independence queries. When r is sublinear in n, the result of [CLSSW19; Ngu19] provides faster (subquadratic) algorithm than that of [LSW15], but for linear r (i.e., $r \approx n$), all of these results are stuck at query complexity of $\tilde{O}(n^2)$. This is known as the *quadratic barrier* [CLSSW19]. A natural question is whether this barrier can be broken [LSW15,

Conjecture 13].

The only previous progress towards breaking this barrier is by [CLSSW19] and falls under the following two categories. Either we need to assume the more powerful *rank* oracle model where [CLSSW19] provides a $\tilde{O}(n^{1.5})$-time algorithm. Or, we solve an *approximate* version of the matroid intersection problem, where [CLSSW19] provides an algorithm with $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ complexity for $(1-\varepsilon)$-approximately solving the matroid intersection problem in the independence oracle model. Breaking the quadratic barrier with an *exact* algorithm in the *independence* query model remains open.

Our results. We break the quadratic barrier with both deterministic and randomized algorithms:

Theorem B.1.1 (Details in Theorems B.4.7 and B.4.8)**.** *Matroid Intersection can be solved by*

- *a deterministic algorithm taking $\tilde{O}(n^{11/6})$ independence queries, and*
- *a randomized (Las Vegas) algorithm taking $\tilde{O}(n^{9/5})$ independence queries with high probability.*

By high probability, we mean probability of at least $1-1/n^c$ for an arbitrarily large constant c. While we only focus on the query complexity in this paper, we note that the time complexities of our algorithms are dominated by the independence oracle queries. That is, our deterministic and randomized algorithms have time complexity $\tilde{O}(n^{11/6}\mathcal{T}_{\text{ind}})$ and $\tilde{O}(n^{9/5}\mathcal{T}_{\text{ind}})$ respectively, where $\mathcal{T}_{\text{ind}}$ denotes the maximum time taken by an oracle to answer an independence query.

Technical overview. Below we explain the key insights of our algorithms which are fast algorithms to solve a graph problem called *reachability* and a simple way to combine our algorithms with the existing exact and approximation algorithms to break the quadratic barrier.

Reachability problem: In this problem, there is a directed bipartite graph G on n vertices with bi-partition $(S \cup \{s,t\}, \bar{S})$. We want to determine whether a directed (s,t)-path exists in G. We know the vertices of G, but not the edge set E of G. We are allowed to ask the following two types of *neighborhood queries*:

1. Out-neighbor query: Given $v \in \bar{S}$ and $X \subseteq S \cup \{s,t\}$, does there exists an edge from v to some vertex in X?
2. In-neighbor query: Given $v \in \bar{S}$ and $X \subseteq S \cup \{s,t\}$, does there exists an edge from some vertex in X to v?

In other words, we can ask an oracle if a "right vertex" $v \in \bar{S}$ has an edge to or from a set X of "left vertices". This problem arose as a subroutine of previous matroid intersection algorithms that are based on finding augmenting paths [AD71; Law75; Cun86; CLSSW19; Ngu19]. Naively, we can solve this problem with quadratic

($O(n^2)$) queries: find all edges of G by making a query for all possible $O(n^2)$ pairs of vertices. Cunningham [Cun86] used this algorithm in his framework to solve the matroid intersection problem with $O(nr^{1.5})$ queries. Recent results by [CLSSW19; Ngu19] solved the reachability problem with $\tilde{O}(nd)$ queries, where d is the distance between s and t in G, essentially by simulating the breadth-first search process. Plugging these algorithms into Cunningham's framework leads to algorithms for the matroid intersection with $\tilde{O}(nr)$ queries. When d is large, the algorithms of [CLSSW19; Ngu19] still need $\tilde{\Theta}(n^2)$ queries to solve the reachability problem. It is not clear how to solve this problem with a subquadratic number of queries. The key component of our algorithms is subquadratic-query algorithms for the reachability problem:

Theorem B.1.2 (Details in Theorems B.3.1 and B.3.2)**.** *The reachability problem can be solved by*

- *a deterministic algorithm that takes $\tilde{O}(n^{5/3})$ queries, and*
- *a randomized (Las Vegas) algorithm that takes $\tilde{O}(n\sqrt{n})$ queries with high probability.*

Plugging Theorem B.1.2 into standard frameworks such as Cunningham's does not directly lead us to a subquadratic-query algorithm for matroid intersection. Our second insight is a simple way to combine algorithms for the reachability problem with the exact and approximation algorithms of [CLSSW19] to achieve the following theorem.

Theorem B.1.3 (Details in Lemma B.4.6)**.** *If there is an algorithm $\mathcal{A}$ that solves the reachability problem with $\mathcal{T}$ queries, then there is an algorithm $\mathcal{B}$ that solves the matroid intersection problem with $\tilde{O}(n^{9/5} + n\sqrt{\mathcal{T}})$ independence queries. If $\mathcal{A}$ is deterministic, then $\mathcal{B}$ is also deterministic.*

Theorems B.1.2 and B.1.3 immediately lead to Theorem B.1.1. We provide proof ideas of Theorems B.1.2 and B.1.3 in the subsections below.

B.1.1 Proof idea for Theorem B.1.2: Algorithm for the reachability problem

Before mentioning an overview of the algorithm for solving the reachability problem, we briefly mention what makes this problem hard. Note that if we discover that some $v \in \bar{S}$ is reachable from s, we can find all out-neighbors of v in $(S \cup \{s, t\})$ in $O(\log n)$ queries per such neighbor. We do this by using a binary search with out-neighbor queries, halving the size of the set of potential out-neighbors of v in each step. However, when we discover that some $v \in S$ is reachable from s, we cannot use the same binary-search trick to efficiently find the out-neighbors of v due to the *asymmetry* of the allowed queries, where we can make queries only

for vertices $v \in \bar{S}$. Such asymmetry makes it hard to efficiently apply a standard (s,t)-reachability algorithm (such as breadth-first search) on the graph.[2]

Both our randomized and deterministic algorithms for the reachability problem follow the same framework below, where we partition vertices in $\bar{S}$ into *heavy and light* vertices and find vertices that can reach some heavy vertices. Our randomized and deterministic algorithms differ in how they determine whether a vertex is heavy or light.

Heavy/Light vertices. Our reachability algorithms run in phases and keep track of a set of vertices that are reachable from the source vertex s, denoted by F (for "found"). We can assume that F *contains all out-neighbors of vertices in* $F \cap \bar{S}$, because we can find these out-neighbors very efficiently by doing binary-search that makes $\tilde{O}(1)$ queries per out-neighbor. In each phase, the algorithm either

(a) increases the size of F by an additive factor of at least h for some parameter h (we use either $h = \sqrt{n}$ or $h = n^{1/3}$), or
(b) returns whether there is an (s,t) path.

Hence, in total, there are at most $\frac{n}{h}$ many phases. To this end, for every $v \in \bar{S}$, we say that v is F*-heavy* if either

(h1) v has at least h out-neighbors to $S \setminus F$, or
(h2) there is an edge from v to t.

If $v \in \bar{S}$ is not F-heavy, we say that it is F*-light*. We omit F when it is clear from the context. We emphasize that the notion of heavy and light applies only to vertices in $\bar{S}$.Two tasks that remain are how to determine if a vertex is heavy or light, and how to use this to achieve (a) or (b).

Heavy vertex reachability. First, we show how to achieve (a) or (b). We assume for now that we know which vertices in $\bar{S}$ are heavy or light. We can also assume that we know all out-going edges of all light vertices (e.g. black edges in Figure B.1); this requires $\tilde{O}(nh)$ queries over all phases. Our main component is to determine a set of vertices that can reach *some* heavy vertex. (Heavy vertices are always in such set.) We can do this with $\tilde{O}(n)$ queries essentially by simulating a breadth-first search process *reversely* from heavy vertices. This process leaves us with subtrees rooted at the heavy vertices with edges pointed to the roots; see Figure B.1 for an example. The actual algorithm is quite simple and can be found in Section B.3.

Once all vertices that can reach some heavy vertices are found, we end up in one of the following situations:

[2]In contrast, in the *symmetric* case where in- and out-neighbor queries can be made for *every* vertex (and not just $v \in \bar{S}$), we can solve the reachability problem with $\tilde{O}(n)$ queries. This requires a simple breadth-first search starting from s where we discover neighbors using binary search.

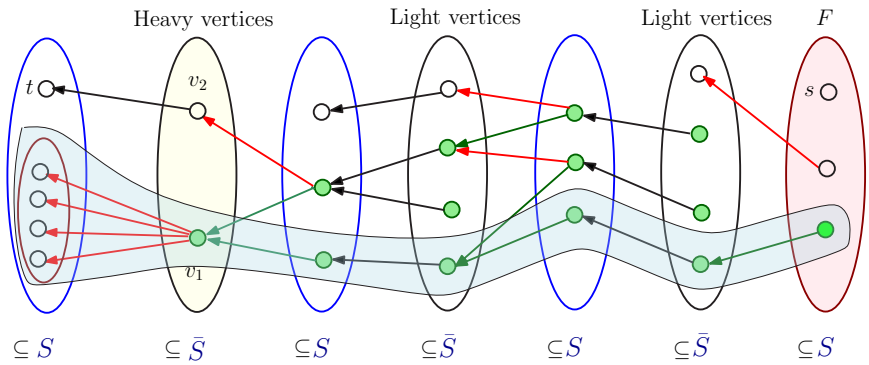


Figure B.1: Example of the reverse BFS process to compute heavy vertex reachability. The vertices from S and $\bar{S}$ occur at alternate layers. The black edges (out-edges of light nodes) are known a priori. The green edges are traversed in the reverse BFS procedure whereas the red edges are not traversed in the reverse BFS. The green vertices are discovered in the reverse BFS. The green vertices form a tree rooted at v_1. Vertex v_1 is heavy because of its large out-degree. Vertex v_2 is heavy because t is its out-neighbor. The path from F to v_1 and the out-neighbors of v_1 are highlighted in light-blue, which is added to F after the reverse BFS. Note that, even though v_2 is reachable from F, the path from F to v_2 is not discovered, and hence the algorithm moves to the next iteration.

- Some vertex in F can reach a heavy vertex v satisfying (h2). In this case, we know immediately that s can reach t via v.
- Some vertex in F can reach a heavy vertex v satisfying (h1). In this case, we query and add all out-neighbors of v in $S \setminus F$ (taking $\tilde{O}(n)$ queries). This adds at least h vertices to F as desired.
- No vertices in F can reach any heavy vertex. In this case, we conclude that s does not reach t: to be able to reach t, s must be able to reach some vertex that points to t (and thus is heavy).

Heavy/light categorization. Again, this is where our randomized and deterministic algorithms differ. With randomness, we can use random sampling to approximate the out-degree of every vertex in $\bar{S}$ and find all out-going edges of vertices that are potentially light. This takes $\tilde{O}(nh + n^2/h)$ queries over all phases. For the deterministic algorithm, a naive idea is to maintain, for every $v \in \bar{S}$, up to h out-going neighbors of v in $S \setminus F$. The challenge is that when these neighbors are included in F, we have to find new neighbors. By carefully picking these neighbors, we can argue that in total only $\tilde{O}(n\sqrt{nh})$ queries are needed over all phases.

Summary. In total, in addition to the categorization of *heavy/light* vertices, we use $\tilde{O}(n)$ queries to solve the heavy vertex reachability problem in each of the $\tilde{O}(n/h)$

phases. We also need $\tilde{O}(nh)$ queries over all phases to find at most h out-neighbors in $S \setminus F$ of light vertices. So, in total, our algorithm uses $\tilde{O}(\frac{n^2}{h} + nh)$ queries plus the number of queries needed for the categorization, which is $\tilde{O}(\frac{n^2}{h} + nh)$ for randomized and $\tilde{O}(n\sqrt{nh})$ for deterministic algorithms.

B.1.2 Proof idea of Theorem B.1.3: From reachability to matroid intersection

The standard connection between the matroid intersection problem and the reachability problem that is exploited by most combinatorial algorithms [AD71; Law75; Cun86; CLSSW19; Ngu19] is based on finding augmenting paths in what is called the *exchange graph*. Given a common independent set S of the two matroids over common ground set V, the *exchange graph* $G(S)$ is a directed bipartite graph over vertex set $V \cup \{s, t\}$ as in the reachability problem above with $\bar{S} = V \setminus S$. The edges of the exchange graph are defined to ensure the following property: Finding an (s, t)-path in the exchange graph amounts to augmenting S, i.e. finding a new common independent with a bigger size. Conversely, if no (s, t)-path exists in the exchange graph, it is known that S is of maximum cardinality and, hence, S can be output as the answer to the matroid intersection problem. Thus the problem of *augmentation* in the exchange graph can be reduced to the *reachability* problem where the neighborhood queries in the reachability problem correspond to the queries to the matroid oracles.[3]

Let us suppose that we can solve the *reachability* problem using $\mathcal{T}$ queries. An immediate and straightforward way of using this subroutine to solve matroid intersection is the following: Call this subroutine iteratively to find augmenting paths to augment along in the exchange graph, thereby increasing the size of the common independent set by one in each iteration. As the size of the largest common independent set is r, we need to perform r augmentations in total. This leads to an algorithm solving matroid intersection using $O(r\mathcal{T})$ independence queries.

To improve upon this, we avoid doing the majority of the augmentations by starting with a good approximation of the largest common independent set. We use the recent subquadratic $(1 - \varepsilon)$-approximation algorithm of [CLSSW19, Section 6] that uses $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ independence queries to obtain a common independent set of size at least $r - \varepsilon r$. Once we obtain a common independent set with such approximation guarantee, we only need to perform an additional εr augmentations. This is still not good enough to obtain a subquadratic matroid intersection algorithm when combined with our efficient algorithms for the reachability problem from Theorem B.1.2.

The final observation we make, is that for small $\varepsilon = o(n^{-1/5})$, we can combine the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ approximation algorithm of [CLSSW19] with an efficient implementation of Cunningham's algorithm (as in [CLSSW19; Ngu19]) to obtain

[3]The independence queries are more powerful than the neighborhood queries, but we are only interested in the neighborhood queries in our algorithm for the reachability problem.

a $(1-\varepsilon)$-approximation algorithm for matroid intersection using $\tilde{O}(n^{9/5}+n/\varepsilon)$ queries. This has a slightly better complexity than just running the approximation algorithm of [CLSSW19]. The idea is to first run the the $\tilde{O}(n^{1.5}/\varepsilon'^{1.5})$ approximation algorithm with $\varepsilon' \approx n^{-1/5}$, and then run the Cunningham-style algorithm until the distance between s and t in the exchange graph becomes at least $\Theta(1/\varepsilon)$.

Our final algorithm is then:

1. Run the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\varepsilon)$-approximation algorithm from [CLSSW19, Section 6] with $\varepsilon = n^{-1/5}$ to obtain a common independent set S of size at least $r - n^{4/5}$. This step takes $\tilde{O}(n^{9/5})$ queries.
2. Starting with S, run the Cunningham-style algorithm as implemented by [CLSSW19, Section 5] until the (s,t)-distance is at least $\sqrt{\mathcal{T}}$ to obtain a common independent set of size at least $r - O(n/\sqrt{\mathcal{T}})$. This step takes $\tilde{O}(n(r-|S|)+n\sqrt{\mathcal{T}}) = \tilde{O}(n^{9/5}+n\sqrt{\mathcal{T}})$ queries.
3. For the remaining $O(n/\sqrt{\mathcal{T}})$ augmentations, find augmenting paths one by one by solving the reachability problem. This step takes $\tilde{O}(n\sqrt{\mathcal{T}})$ queries.

Hence we obtain a matroid intersection algorithm which uses $\tilde{O}(n^{9/5}+n\sqrt{\mathcal{T}})$ independence queries, as in Theorem B.1.3.

B.1.3 Organization

We start with the necessary preliminaries in Section B.2. In Section B.3, we provide the subquadratic deterministic and randomized algorithms for augmentation. Finally, in Section B.4, we combine these algorithms for augmentation with existing algorithms to obtain subquadratic deterministic and randomized algorithms for matroid intersection. In Section B.3, we skip the description of an important subroutine called the heavy/light categorization. We devote Section B.5 for details of this subroutine.

B.2 Preliminaries

Matroid. A *matroid* is a combinatorial object defined by the tuple $\mathcal{M} = (V, \mathcal{I})$, where the ground set V is a finite set of elements and $\mathcal{I} \subseteq 2^V$ is a non-empty family of subsets (denoted as the *independent sets*) of the ground set V, such that the following properties hold:

1. **Downward closure:** If $S \in \mathcal{I}$, then any subset $S' \subset S$ (including the empty set) is also in $\mathcal{I}$,

2. **Exchange property:** For any two sets $S_1, S_2 \in \mathcal{I}$ with $|S_1| < |S_2|$, there is an element $v \in S_2 \setminus S_1$ such that $S_1 \cup \{v\} \in \mathcal{I}$.

Matroid Intersection. Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V, the *matroid intersection problem* is finding a maximum cardinality common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. When discussing matroid intersection, we will denote by r the size of such a maximum cardinality common independent set and by n the size of the ground set V.

Exchange graph. Consider two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V. Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a common independent set. The *exchange graph* $G(S)$, w.r.t. to the common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, is defined to be a directed bipartite graph where the two sides of the bipartition are S and $\bar{S} = V \setminus S$. Moreover, there are two additional special vertices s and t (that are not included in either S or $\bar{S}$) which have directed edges incident on them *only* from $\bar{S}$. The directed edges (or arcs) are interpreted as follows:

1. Any edge of the form (s, v) for $v \in \bar{S}$ implies that $S \cup \{v\}$ is an independent set in $\mathcal{M}_1$.

2. Similarly, any edge of the form (v, t) for $v \in \bar{S}$ implies that $S \cup \{v\}$ is an independent set in $\mathcal{M}_2$.

3. Any edge of the form $(u, v) \in S \times \bar{S}$ implies that $(S \setminus \{u\}) \cup \{v\}$ is an independent set in $\mathcal{M}_1$.

4. Similarly, any edge of the form $(v, u) \in \bar{S} \times S$ implies that $(S \setminus \{u\}) \cup \{v\}$ is an independent set in $\mathcal{M}_2$.

We are interested in the notion of *chordless* (s, t)-paths in $G(S)$ [Cun86, Section 2] which are defined next. For this definition, we consider a path as a sequence of vertices that take part in the path. A subsequence of a path is an ordered subset of the vertices (not necessarily contiguous) of the path where the ordering respects the path ordering.

Definition B.2.1. An (s, t)-path p is *chordless* if there is no proper subsequence of p which is also an (s, t)-path. A chordless path in the exchange graph $G(S)$ is sometimes called an *augmenting path*.

Claim B.2.2 (Augmenting path)**.** *Consider a chordless path p from s to t in $G(S)$ (if it exists), and let $V(p)$ be the elements of the ground set (or, equivalently, vertices in the exchange graph excluding s and t) that take part in the path p. Then $S \triangle V(p)$ is a common independent set of $\mathcal{M}_1$ and $\mathcal{M}_2$.*

If we examine the set $S \triangle V(p)$ obtained from Claim B.2.2, it is clear that the number of elements added to the set S is one more than the number of elements removed from S. This observation immediately gives the following corollary, and shows the importance of the notion of exchange graphs.

Corollary B.2.3. *The size of the largest common independent set of $\mathcal{M}_1$ and $\mathcal{M}_2$ is at least $|S|+1$ if and only if t is reachable from s in $G(S)$.*

It is useful to note that the shortest (s,t)-path in $G(S)$ is always chordless. Many combinatorial matroid intersection algorithms thus focus on finding shortest (s,t)-paths. The following claim relating the distance from s to t in $G(S)$ and the size of S is useful for approximation algorithms for matroid intersection.

Claim B.2.4 ([Cun86])**.** *If the length of the shortest (s,t)-path in $G(S)$ is at least d, then $|S| \geq (1-O(\frac{1}{d}))r$, where r is the size of the largest common independent set.*

Matroid query oracles. There are two primary models of query oracles associated with the matroid theory: (i) the independence query oracle, and (ii) the rank query oracle. The independence query oracle, given a set $S \subseteq V$ of a matroid $\mathcal{M}$, outputs 1 iff S is an independent set of $\mathcal{M}$ (i.e., iff $S \in \mathcal{I}$). The rank query oracle, given a set $S \subseteq V$, outputs the rank of S, $\text{rank}_{\mathcal{M}}(S) \stackrel{def}{=} \max_{T \subseteq S: T \in \mathcal{I}} |T|$, i.e., the size of the largest independent set contained in S. Clearly, if S itself is an independent set, then $\text{rank}_{\mathcal{M}}(S) = |S|$. Hence, a rank query oracle is at least as powerful as the independence query oracle. In this work, we are however interested primarily in the independence query oracle model. Next, we state two claims regarding the independence query oracle that we use in the paper.

Claim B.2.5 (Edge discovery)**.** *By issuing one independence query each, we can find out*

(i) given a vertex $v \in \bar{S}$, whether v is an out-neighbor of s; or

(ii) given a vertex $v \in \bar{S}$, whether v is an in-neighbor of t; or

(iii) given a vertex $v \in \bar{S}$ and a subset $X \subseteq S$, whether there exists an edge from some vertex in X to v; or

(iv) given a vertex $v \in \bar{S}$ and a subset $X \subseteq S$, whether there exists an edge from v to some vertex in X.

Claim B.2.5 follows from observing that we can make the following kinds of independence queries: (i-ii) whether $S \cup \{v\}$ is an independent set in $\mathcal{M}_1$ respectively $\mathcal{M}_2$, and (iii-iv) whether $S \cup \{v\} \setminus X$ is an independent set in $\mathcal{M}_1$ respectively $\mathcal{M}_2$. Note that these edge-discovery queries can simulate the neighborhood-queries in the reachability problem.

With these kinds of queries, we can perform a binary search to find an in-/out-neighbor of $v \in \bar{S}$. The following lemma is proven in [CLSSW19, Lemma 11] and also mentioned in [Ngu19]. We skip the proof in the paper.

Claim B.2.6 (Binary search with independence/neighborhood queries, [Ngu19; CLSSW19])**.** *Consider a vertex $v \in \bar{S}$ and a subset $X \subseteq S \cup \{s,t\}$. By issuing $O(\log r)$ independence queries to $\mathcal{M}_1$, we can find a vertex $u \in X$ such that there is an edge (u,v) (i.e., u is an in-neighbor of v), or otherwise determine that no such edge exists. Similarly, by issuing $O(\log r)$ independence queries to $\mathcal{M}_2$, we can find a vertex $u' \in X$ such that there is an edge (v,u') (i.e., u' is an out-neighbor of v).*

We will assume InEdge(v,X) respectively OutEdge(v,X) are procedures which implement Claim B.2.6.

B.3 Algorithms for augmentation

From Claim B.2.2, we know the following: Given a common independent set S, either S is of maximum cardinality or there exists a (directed) (s,t)-path in the exchange graph $G(S)$. In this section, we consider the (s,t)-reachability problem in $G(S)$ using independence oracles. Our main results in this section are the following two theorems. We denote the size of S as $|S| = r$ in both of these theorems.[4]

Theorem B.3.1 (Randomized augmentation)**.** *There is a randomized algorithm which with high probability uses $O(n\sqrt{r}\log n)$ independence queries and either determines that S is of maximum cardinality or finds an augmenting path in $G(S)$.*

Theorem B.3.2 (Deterministic augmentation)**.** *There is a deterministic algorithm which uses $O(nr^{2/3}\log r)$ independence queries and either determines that S is of maximum cardinality or finds an augmenting path in $G(S)$.*

B.3.1 Overview of the algorithms

Section B.1.1 gives an informal overview of the augmentation algorithm already. In this section, we provide more details so that the reader can be convinced about the correctness of the algorithm.

The algorithm for augmentation, denoted as Augmentation algorithm for easy reference, runs in phases and keeps track of a set F of vertices that are reachable from the vertex s. Let F_S and $F_{\bar{S}}$ denote the bipartition of F inside S and $\bar{S}$, i.e., $F_S = F \cap S$ and $F_{\bar{S}} = F \cap \bar{S}$. In each phase, the algorithm will increase the size of F_S by an additive factor of at least h until the algorithm discovers an (s,t)-path (or, otherwise, discover there is no such path). Hence, in total, there are at most $\frac{|S|}{h}$ many phases. We now give an overview of how to implement each phase.

Note that, without loss of generality, we can assume that the set F_S contains all vertices that are out-neighbors of vertices in $F_{\bar{S}}$. This is because whenever a vertex $v \in \bar{S}$ is added to $F_{\bar{S}}$, we can quickly add all of v's out-neighbors in $S \setminus F_S$ into the set F_S by using Claim B.2.6. This requires $O(\log r)$ independence queries

[4] Note that r usually denotes the size of the maximum common independent set which is an upper bound on the size of the vertex set S. We abuse the notation and use r here to denote $|S|$.

for each such out-neighbor. Hence, in total, this procedure uses at most $O(n \log r)$ independence queries, since each $u \in S \setminus F_S$ is added in F_S at most once.

Heavy and light vertices. Before explaining what the algorithm does in each phase, we introduce the notion of *heavy* and *light* vertices: We divide the vertices in $\bar{S} \setminus F_{\bar{S}}$ into two categories. We call a vertex $v \in \bar{S} \setminus F_{\bar{S}}$ *heavy* if it either has an edge to t or has at least h out-neighbors in $S \setminus F_S$. The vertices in $\bar{S} \setminus F_{\bar{S}}$ that are not *heavy* are denoted as *light* (See Figure B.1 for reference; the heavy nodes are highlighted in light-yellow). Note that both these notions are defined in terms of out-degrees, i.e., a heavy vertex can have arbitrary in-degree and so can a light vertex. Also, note that the notion of heavy and light vertices are defined w.r.t. to the set F_S. Because the set F_S changes from one phase to the next, so does the set of heavy vertices and light vertices.

Description of phase i. Let us assume, for the time being, that there is an efficient procedure to categorize the vertices in $\bar{S} \setminus F_{\bar{S}}$ into the sets of heavy and light vertices. We first apply this procedure at the beginning of phase i.

Now, for simplicity, consider an easy case: In phase i, there is a *heavy* vertex that has an in-neighbor in F_S. In this case, we can go over all vertices in $\bar{S} \setminus F_{\bar{S}}$ to find such a heavy vertex—this can be done with n many independence queries. Once we find such a heavy vertex, we include it in F_S and all of its out-neighbors in $F_{\bar{S}}$. Note that, in this case, either of the following two things can happen: either we have increased the size of F_S by at least h as the heavy vertex has at least h out-neighbors in $S \setminus F_S$; or the heavy vertex we found has t as its out-neighbor in which case we have found an (s, t)-path.

Unfortunately, this may not be the case in phase i. In this case, we do an additional procedure called the *reverse breadth-first search* or, in short, *reverse BFS*. The goal of the reverse BFS is to find a heavy vertex reachable from F. Before describing this procedure, note the following two properties of the *light* vertices:

1. A light vertex will remain a light vertex even if we increase the size of F_S.

2. We can assume that we know all out-neighbors of any light vertex.

Property 2 needs some explanation. This property is true because of two observations: (i) All out-neighbors of a light vertex can be found out with $O(h \log n)$ independence queries using Claim B.2.6, and (ii) because of Property 1, across all phases, we need to find out the out-neighbors of a light vertex *only once*. So, even though we need to make $O(nh \log n)$ queries in total, this cost amortizes across all phases.

The idea is, as before, to discover a heavy vertex which is reachable from F so that we can include all of its out-neighbors in F_S (for example, consider the heavy vertex v_1 in Figure B.1). So our goal is to find some path from F to a heavy vertex (Consider the path starting from v_1 highlighted in light-blue in Figure B.1). This naturally implies the need for doing a reverse BFS from the heavy vertices. We also

note that any path from F to t must pass through a heavy vertex (the vertex just preceding t must by definition be heavy). Hence, if our reverse BFS fails to find a path from F to some heavy vertex, the algorithm has determined that no (s,t)-path exists.

What remains is to find out how to implement the reverse BFS procedure efficiently. To this end, we exploit Property 2 of light vertices and assume that we know all edges directed from $\bar{S}$ to S that the reverse BFS procedure needs to visit. This follows from the following crucial observation: *No internal node of the reverse BFS forest is a heavy node*, i.e., in other words, the heavy vertices occur *only* as root nodes of the reverse BFS trees. This is because if, along the traversal of a reverse BFS procedure starting from a heavy node v, we reach another heavy node v', we can ignore v' as the reverse BFS starting from node v' has already taken care of processing v'. This means that any edge in $\bar{S} \times S$ that takes part in the reverse BFS procedure must originate from a light vertex and, hence, is known a priori due to Property 2. All it remains for the reverse BFS procedure is to discover in-neighbors of vertices in $\bar{S}$ using edges from $S \times \bar{S}$. By Claim B.2.6, each such in-neighbor can be found by making $O(\log r)$ independence queries. In total, the reverse BFS procedure uses $\tilde{O}(n)$ independence queries.

Post-processing. Note that, in order to use Claim B.2.2, the (s,t)-path needs to be chordless. However, the (s,t)-path p that the algorithm outputs has no such guarantee. So, as a post-processing step, the algorithm uses an additional $\tilde{O}(r)$ independence queries to convert this path into a *chordless* path: Consider any vertex $v \in V(p) \cap \bar{S}$ and assume u as the parent of v, and w as the child of v in the path p. The vertex v needs to check whether it has an in-neighbor other than u among the ancestors of v in $V(p)$ or an out-neighbor other than w among the descendants of v in $V(p)$. Since the length of the path obtained from the previous step is $O(r)$ (because of $|S| = r$ and the path does not contain any cycle), this requires $O(\log r)$ independence queries. If all vertices in $V(p) \cap \bar{S}$ have no such in or out-neighbors, then it is easy to see that p is indeed a chordless path. If there is such a (say) in-neighbor u' of v, then we remove all vertices of $V(p)$ between u' and v, and the resulting subsequence is still an (s,t)-path. A similar procedure is done when an out-neighbor is discovered. In total, this takes $O(r \log r)$ independence queries, since each vertex can be removed from the path at most once.

Cost analysis. The total number of queries needed to implement phase i is a summation of two terms: (i) the number of queries needed to partition the vertices into heavy and light categories, and (ii) the number of queries needed to run the reverse BFS procedure. We have seen that (ii) can be implemented with $\tilde{O}(n)$ independence queries. For (i), we present two algorithms: a randomized sampling algorithm, and a deterministic algorithm which is slightly less efficient than the randomized one. This is the main technical difference between the algorithm of Theorem B.3.1 and that of Theorem B.3.2. The cost analysis for (i) is also amortized

and the total number of queries needed across all phases is $\tilde{O}(\max\{nh, nr/h\})$ for randomized and $\tilde{O}(n\sqrt{rh})$ for deterministic implementation. Setting $h = \sqrt{r}$ for randomized and $h = r^{1/3}$ for deterministic, we see that total randomized query complexity of augmentation is $\tilde{O}(n\sqrt{r})$ and deterministic query complexity is $\tilde{O}(nr^{2/3})$.

B.3.2 Categorizing *heavy* and *light* vertices

We start with reminding the readers the definition of the *heavy* and *light* vertices in $\bar{S} \setminus F_{\bar{S}}$.

Definition B.3.3. We call a vertex $v \in \bar{S} \setminus F_{\bar{S}}$ heavy if either (v, t) is an edge of $G(S)$ or v has at least h out-neighbors in $S \setminus F_S$. Otherwise we call v light.

To check whether v has an edge to t is easy and requires only a single independence query: "Is $S \cup \{v\}$ independent in $\mathcal{M}_2$?" The difficulty lies when this is not the case and we need to determine if v has outdegree at least h to $S \setminus F_S$. We present two algorithms to solve this categorization problem: one randomized sampling algorithm; and a less efficient deterministic algorithm. More concretely, we show the following two lemmas.

Lemma B.3.4. *There is a randomized categorization procedure which, with high probability, categorizes heavy and light vertices in the set $\bar{S} \setminus F_{\bar{S}}$ correctly by issuing $O(n \log n)$ independence queries per phase and an additional $O(nh \log n)$ independence queries over the whole run of the* Augmentation *algorithm.*

Lemma B.3.5. *There exists a deterministic categorization procedure which uses $O(n\sqrt{rh} \log r)$ queries over the whole run of the* Augmentation *algorithm.*

The proofs of these two lemmas are deferred to Section B.5.

B.3.3 Heavy vertex reachability

In this section, we present the reverse BFS in Algorithm B.1 and analyze some properties of it. Recall that the reverse BFS is run once in each phase of the algorithm to find some vertex in F which can reach some heavy vertex. We also remind the reader of the example in Figure B.1. In this section, we prove the following.

Lemma B.3.6 (Heavy vertex reachability)**.** *There is an algorithm (Algorithm B.1: ReverseBFS) which, given F such that there are no edges from $F_{\bar{S}}$ to $S \setminus F_S$, a categorization of $\bar{S} \setminus F_{\bar{S}}$ into* heavy *and* light*, and all out-edges of the light vertices to $S \setminus F_S$, uses $O(n \log r)$ queries and either finds a path from some vertex in F to a heavy node, or otherwise determines that no such path exists.*

We next provide the pseudo-code (Algorithm B.1).

Algorithm B.1: ReverseBFS

Input: Categorization of $\bar{S} \setminus F_{\bar{S}}$ into *heavy* and *light*; and a set LightEdges containing all out-edges of the light vertices
Output: A path from F to some heavy vertex, if one exists

```
1  Q ← {v ∈ S̄ \ F_S̄ which are heavy}
2  NotVisited ← (S ∪ S̄ ∪ {s, t}) \ Q
3  while Q ≠ ∅ do
4      Pop a vertex v from Q
5      if v ∈ F then
6          return the path from v to a heavy vertex in the BFS-forest
7      else if v ∈ S̄ \ F_S̄ then
8          while u = InEdge(v, NotVisited) is not ∅ do
9              Push u to Q and remove it from NotVisited
10     else if v ∈ S \ F_S then
11         foreach u ∈ NotVisited such that (u, v) ∈ LightEdges do
12             Push u to Q and remove it from NotVisited
13 return "NO PATH EXISTS"
```

Correctness. We first argue that the algorithm is correct. When a vertex $v \in \bar{S} \setminus F_{\bar{S}}$ is processed by the algorithm, each unvisited in-neighbor will be added to the queue Q in the while loop in line 8. When a vertex $v \in S \setminus F_S$ is processed by the algorithm, any edge from NotVisited to v must originate from a light vertex, since NotVisited contains no heavy vertices and we are guaranteed that no edge from $F_{\bar{S}}$ to $S \setminus F_S$ exist. Hence Algorithm B.1 will eventually process every vertex reachable, by traversing edges in reverse, from the heavy vertices.

Cost analysis. The only place Algorithm B.1 uses independence queries is in line 8. Each vertex will be discovered at most once by the binary search in InEdge. This means that we do at most n calls to InEdge, each using $O(\log r)$ queries by Claim B.2.6. Hence the reverse BFS uses $O(n \log r)$ queries per phase.

B.3.4 Augmenting path algorithm.

We now present the main augmenting path algorithm, as explained in the overview in Section B.3.1.

Note that we have not specified if we are using the randomized or deterministic categorization of heavy and light vertices, from Sections B.5.1 and B.5.2. We will for now assume this categorization procedure as a black box which is always correct.

We start by stating some invariants of Algorithm B.2.

Algorithm B.2: AUGMENTATION

Input: Two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$, and a common independent set $S \subseteq \mathcal{I}_1 \cap \mathcal{I}_2$

Output: An augmenting (s,t)-path in $G(S)$ if one exists

```
1  F ← {s}
2  LightEdges ← ∅
3  while t ∉ F do
       // Description of a phase
4      Categorize v ∈ S̄ \ F_S̄ into heavy and light
5      foreach new light vertex v do
6          Use OutEdge to find all out-neighbors of v in S \ F_S
7          Add edges (v, u) to LightEdges for each such out-neighbor u
8      p ← ReverseBFS(S, F, LightEdges)
9      if p = "NO PATH EXISTS" then
10         return "NO PATH EXISTS"
11     else
12         Denote by V(p) the vertices on the path p
13         Add all v ∈ V(p) to F
14         foreach v ∈ V(p) ∩ S̄ do
15             while u = OutEdge(v, (S \ F) ∪ {t}) is not ∅ do
16                 Add u to F
   // Post-processing
17 Post-process the (s,t)-path found to make it chordless
18 return the augmenting path
```

1. F contains only vertices reachable from s. In fact, for each vertex in F we have found a path from s to this vertex.

2. LightEdges contains all out-edges from light vertices to $S \setminus F$.

3. In the beginning of each phase, there exists no $v \in F$, $u \in (S \cup \{s,t\}) \setminus F$ such that (v,u) is an edge in $G(S)$. This is because whenever $v \in \bar{S}$ is added to F, all v's neighbors are also added, see line 15.

Correctness. When the algorithm outputs an (s,t)-path, the path clearly exists, by Invariant 1. So it suffices to argue that the algorithm does not return "NO PATH EXISTS" incorrectly. Note that the algorithm only returns "NO PATH EXISTS" when ReverseBFS does so, that is when there is no path from F to a heavy vertex (by Lemma B.3.6). So suppose that this is that case, and also suppose, for the sake of a contradiction, that an (s,t)-path p exists in $G(S)$. Denote by v the vertex preceding t in the path p. By Invariant 3 we know that v is not in F. But then v is

heavy, since (v, t) is an edge of $G(S)$. Hence a subpath of p will be a path from F to the heavy vertex v, which is the desired contradiction.

Number of phases. We argue that there are at most $\frac{r}{h}+1$ phases of the algorithm. After a phase, either the algorithm returns "NO PATH EXISTS" (in which case this was the last phase), or some path p was found by the reverse BFS. Then $V(p)$ must include some heavy vertex v. Then all neighbors of v will be added to F in line 15. Thus we know that either t was added to F (in which case this was the last phase), or at least h vertices from S was added to F. Since $|S| \leq r$ in the beginning, this can happen at most $\frac{r}{h}$ times.

Number of queries. We analyse the number of independence queries used by different parts of the algorithm:

- ReverseBFS (Algorithm B.1) is run once each phase, and uses $O(n \log r)$ queries per call by Lemma B.3.6. This contributes a total of $O(\frac{nr \log r}{h})$ independence queries over all phases.
- Each $u \in S$ is discovered at most once by the OutEdge call on line 15. So this line contributes a total of $O(n \log r)$ independence queries.
- Each vertex becomes *light* at most once over the run of the algorithm. When this happens, the algorithm finds all of its (up to h) out-neighbors on line 6, using OutEdge calls. This contributes a total of $O(nh \log r)$ independence queries.
- The post-processing can be performed using $O(r \log r)$ independence queries, as explained in Section B.3.1.
- The *heavy*/*light*-categorization uses $O(nh \log n + \frac{nr \log n}{h})$ independence queries when the randomized procedure is used, by Lemma B.3.4. When the deterministic categorization procedure is used, we use $O(n\sqrt{rh} \log r)$ independence queries instead, by Lemma B.3.5.

We see that in total, the algorithm uses:

- $O(n\sqrt{r} \log n)$ independence queries with the randomized categorization, setting $h = r^{1/3}$.
- $O(nr^{2/3} \log r)$ independence queries with the deterministic categorization, setting $h = \sqrt{r}$.

The above analysis proves Theorems B.3.1 and B.3.2.

Remark B.3.7. When the randomized categorization procedure fails, Algorithm B.2 will still always return the correct answer, but it might use more independence queries. So Algorithm B.2 is in fact a Las-Vegas algorithm with expected query-complexity $O(n\sqrt{r} \log n)$.

Remark B.3.8. We note that our algorithm can not be used to find which vertices are reachable from s using subquadratic number of queries.

B.4 Algorithm for fast Matroid Intersection

There are two hurdles to getting a subquadratic algorithm for Matroid Intersection. Firstly, standard augmenting path algorithms need to find the augmenting paths one at a time. This is since after augmenting along a path, the edges in the exchange graph change (some edges are added, some removed). This is unlike bipartite matching, where a set of vertex-disjoint augmenting paths can be augmented along in parallel. It is not clear how to find the augmenting paths faster than $\Theta(n)$ each, so these standard augmenting path algorithms are stuck at $\Omega(nr)$ independence queries.

To overcome this, Chakrabarty-Lee-Sidford-Singla-Wong [CLSSW19] introduce the notion of *augmenting sets*, which allows multiple parallel augmentations. Using the augmenting sets they present a subquadratic $(1-\varepsilon)$-approximation algorithm using $\tilde{O}(\frac{n^{1.5}}{\varepsilon^{1.5}})$ independence queries:

Lemma B.4.1 (Approximation algorithm [CLSSW19])**.** *There exists an* $(1-\varepsilon)$ *approximation algorithm for matroid intersection using* $O(\frac{n\sqrt{n\log r}}{\varepsilon\sqrt{\varepsilon}})$ *independence queries.*

The second hurdle is that when the distance d between s and t is high, the breadth-first algorithms of [CLSSW19; Ngu19] use $\tilde{\Theta}(dn)$ independence queries to compute the distance layers, which is $\Omega(nr)$ when $d \approx r$.[5] Here our algorithm from Section B.3 helps since it can find a single augmenting path using a subquadratic number of independence queries, even when the distance d is large.

So our idea is as follows:

- Start by using the subquadratic approximation algorithm. This avoids having to do the majority of augmentations one by one.
- Continue with the fast implementation [CLSSW19, Section 5] (or [Ngu19]) of the Cunningham-style blocking flow algorithm.
- When the (s,t)-distance becomes too large, fall back to using the augmenting-path algorithm from Section B.3 to find the (few) remaining augmenting paths.

The choice of d will be different depending on whether we use the randomized or deterministic version of Algorithm B.2. In order to run Algorithm B.3, we need to know r so that we may choose ε (and d) appropriately. However, the size r

[5] Note that unlike in Section B.3, we now use the normal definition of r as the size of the maximum-cardinality common independent set of the two matroids.

Algorithm B.3: Subquadratic Matroid Intersection

2 Run the approximation algorithm (Lemma B.4.1) with $\varepsilon = n^{1/5} r^{-2/5} \log^{-1/5} r$ to obtain a common independent set S of size at least $(1-\varepsilon)r = r - n^{1/5} r^{3/5} \log^{-1/5} r$

4 Starting with S, run Cunningham's algorithm (as implemented by [CLSSW19]), until the distance between s and t becomes larger than d

6 Keep running AUGMENTATION (Algorithm B.2) from Section B.3 and augmenting the current common independent set with the obtained (s,t)-path (as in Claim B.2.2) until no (s,t)-path can be found in the exchange graph

of the largest common independent set is unknown. We note that it suffices, for the purpose of the asymptotic analysis, to use a $\frac{1}{2}$-approximation $\bar{r}$ for r (that is $\bar{r} \leq r \leq 2\bar{r}$). It is well known that such an $\bar{r}$ can be found in $O(n)$ independence queries by greedily finding a maximal common independent set in the two matroids. Now we can bound the query complexity of Algorithm B.3.

Lemma B.4.2. *Line 2 of Algorithm B.3 uses $O(n^{6/5} r^{3/5} \log^{4/5} r)$ independence queries.*

Proof. The approximation algorithm uses $O(\frac{n^{1.5}\sqrt{\log r}}{\varepsilon^{1.5}}) = O(n^{6/5} r^{3/5} \log^{4/5} r)$ independence queries, when $\varepsilon = n^{1/5} r^{-2/5} \log^{-1/5} r$. □

Lemma B.4.3. *Line 4 of Algorithm B.3 uses $O(n^{6/5} r^{3/5} \log^{4/5} r + nd \log r)$ independence queries.*

Proof. There are two main parts of Cunningham's blocking-flow algorithm.

- Computing the distances. The algorithm will run several BFS's to compute the distances. The total number of independence queries for all of these BFS's can be bounded by $O(dn \log r)$, since the distances are monotonic so each vertex is tried at a specific distance at most once. For more details, see [CLSSW19, Section 5.1].
- Finding the augmenting paths. Given the distance-layers, a single augmenting path can be found in $O(n \log r)$ independence queries, by a simple depth-first-search. Again, we refer to [CLSSW19, Section 5.2] for more details. Since we start with a common independent set S of size $(1-\varepsilon)r = r - n^{1/5} r^{3/5} \log^{-1/5} r$, we know that S can be augmented at most $n^{1/5} r^{3/5} \log^{-1/5} r$ additional times. Hence a total of $O(n^{6/5} r^{3/5} \log^{4/5} r)$ independence queries suffices to find all of these augmenting paths.

□

Remark B.4.4. We note that if we skip Line 6 in Algorithm B.3, we thus get a $(1-\frac{1}{d})$-approximation algorithm (by Claim B.2.4), using $\tilde{O}(n^{6/5}r^{3/5}+nd)$ independence queries, which beats the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ approximation algorithm when $\varepsilon = o(n^{1/5}r^{-2/5})$.

Lemma B.4.5. *Line 6 of Algorithm B.3 uses $O(\frac{r}{d}\mathcal{T})$ independence queries, where $\mathcal{T}$ is the number of independence queries used by one invocation of* Augmentation *(Algorithm B.2).*

Proof. After line 4, the algorithm has found a common independent set of size at least $(1-O(\frac{1}{d}))r = r - O(\frac{r}{d})$, by Claim B.2.4. This means that only $O(\frac{r}{d})$ additional augmentations need to be performed. □

By Lemmas B.4.2, B.4.3 and B.4.5, we see that Algorithm B.3 uses a total of $O(n^{6/5}r^{3/5}\log^{4/5} r + nd\log r + \frac{r}{d}\mathcal{T})$ independence queries. If we pick $d = \sqrt{\frac{r\mathcal{T}}{n\log r}}$ we get the following lemma.

Lemma B.4.6. *If the query complexity of* Augmentation *is $\mathcal{T}$, then matroid intersection can be solved using $O(n^{6/5}r^{3/5}\log^{4/5} r + \sqrt{nr\mathcal{T}\log r})$ independence queries.*

Combining with Theorems B.3.1 and B.3.2 we get our subquadratic results.

Theorem B.4.7 (Randomized Matroid Intersection)**.** *There is a randomized algorithm which with high probability uses $O(n^{6/5}r^{3/5}\log^{4/5} r)$ independence queries and solves the matroid intersection problem. When $r = \Theta(n)$, this is $\tilde{O}(n^{9/5})$.*

Theorem B.4.8 (Deterministic Matroid Intersection)**.** *There is a deterministic algorithm which uses $O(nr^{5/6}\log r + n^{6/5}r^{3/5}\log^{4/5} r)$ independence queries and solves the matroid intersection problem. When $r = \Theta(n)$, this is $\tilde{O}(n^{11/6})$.*

Remark B.4.9. The limiting term for the the randomized algorithm is between line 2 and line 4. If a faster approximation algorithm is found, the same strategy as above might give an $\tilde{O}(nr^{3/4})$-query algorithm.

B.5 Algorithm for heavy/light categorization

In this section, we finally provide the algorithm for the categorization of vertices in $\bar{S} \setminus F_{\bar{S}}$ into heavy and light vertices as defined in Definition B.3.3.

B.5.1 Randomized categorization

In this section, we prove the following lemma (restated from Section B.3.2).

Lemma B.3.4. *There is a randomized categorization procedure which, with high probability, categorizes heavy and light vertices in the set $\bar{S} \setminus F_{\bar{S}}$ correctly by issuing $O(n \log n)$ independence queries per phase and an additional $O(nh \log n)$ independence queries over the whole run of the* Augmentation *algorithm.*

We will use X to denote $S \setminus F$. Let the out-neighborhood of a vertex $v \in \bar{S} \setminus F_{\bar{S}}$ inside X be denoted as $\mathsf{Ngh}_X(v)$. Consider the family of sets $\{\mathsf{Ngh}_X(v)\}_{v \in \bar{S} \setminus F_{\bar{S}}}$ residing inside the ambient universe X. We want to find out which of these sets are of size at least h (i.e., correspond to the heavy vertices) and which of them are not (i.e., corresponds to the light vertices). To this end, we devise the following random experiment.

Experiment B.5.1. *Sample a set R of k elements drawn uniformly and independently from X (with replacement) and check whether $R \cap \mathsf{Ngh}_X(v) = \emptyset$.*

It is easy to check the following: For any $v \in \bar{S} \setminus F_{\bar{S}}$, Experiment B.5.1 is successful with probability:

$$\Pr_R[R \cap \mathsf{Ngh}_X(v) = \emptyset] = \left(1 - \frac{|\mathsf{Ngh}_X(v)|}{|X|}\right)^k.$$

Note that, to perform this experiment for a vertex v, we need to make a single independence query of the form whether $(S \setminus R) \cup \{v\} \in \mathcal{I}_2$. Next, we make the following claim.

Claim B.5.2. *There is a non-negative integer k such that the following holds:*

1. *If $|\mathsf{Ngh}_X(v)| < h$, then Experiment B.5.1 succeeds with probability at least 3/4, and*

2. *If $|\mathsf{Ngh}_X(v)| > 10h$, then Experiment B.5.1 succeeds with probability at most 1/4.*

Before proving Claim B.5.2, we show the rest of the steps of this procedure. For every vertex, we repeat Experiment B.5.1 $s = O(\log n)$ many times independently. By standard concentration bound, we make the following observations:

1. If $|\mathsf{Ngh}_X(v)| < h$, strictly more than $s/2$ experiments succeed with very high probability.[6]

2. If $|\mathsf{Ngh}_X(v)| > 10h$, strictly less than $s/2$ experiments succeed with very high probability.

[6] Recall that by *very high probability* we mean with probability at least $1 - n^{-c}$ for some arbitrary large constant c.

Hence, we declare any vertex for which strictly less than $s/2$ experiments succeed as *heavy*. The probability that a light vertex can be classified as heavy by this procedure is very small due to Property 1. On the other hand, a vertex with $|\mathsf{Ngh}_X(v)| > 10h$ will be correctly classified as heavy with a very high probability. However, a heavy vertex with $|\mathsf{Ngh}_X(v)| \leq 10h$ may not be correctly classified. So, for such vertices, we want to check in a brute-force manner. To this end, we discover the set $\mathsf{Ngh}_X(v)$ for any vertex v which is not declared heavy and make decisions accordingly.

Bounding the error probability. We argue that we can bound the error probabilities from Properties 1 and 2 over the whole run of the AUGMENTATION algorithm by a union bound. Say that the error probabilities of Properties 1 and 2 is bounded by n^{-c} for some large constant $c \geq 10$. In each phase we categorize at most n vertices, and there is at most $\frac{r}{h} < n$ phases. Hence, the probability that — over the whole run of AUGMENTATION (Algorithm B.2) — that any vertex is misclassified as heavy, or that the procedure decides to discover a set $\mathsf{Ngh}_X(v)$ with $\mathsf{Ngh}_X(v) > 10$, is at most n^{-c+2}. Similarly we note that in the algorithm for Matroid Intersection (Algorithm B.3) we run AUGMENTATION at most r times, so the error probability is at most n^{-c+3}.

Cost analysis. As mentioned before, each instance of Experiment B.5.1 can be performed with a single query. As there are $O(n \log n)$ experiments in total in each phase of the algorithm, the number of queries needed to perform all experiments over the whole run of the AUGMENTATION algorithm will be is $O(\frac{nr \log n}{h})$ (recall that the number of phases is r/h). Now consider the part of the algorithm where we need to discover the set $\mathsf{Ngh}_X(v)$ for any vertex v which is not declared heavy after the completion of all experiments in a phase. For each such vertex, this will take at most $O(|\mathsf{Ngh}_X(v)| \log n) = O(h \log n)$ queries (due to Claim B.2.6). Note that we only need to make these kinds of queries from each vertex once over the whole run of the algorithm (as in future queries we already know all v's neighbors and can answer directly). Hence, the total number of such queries is at most $O(nh \log n)$ across all phases of the algorithm.

Proof of Claim B.5.2. First we note that if $|X| \leq 10h$, case 2 is vacuously true, so we may pick $k = 0$ such that Experiment B.5.1 always succeeds. So now assume that $|X| > 10h$ and let $x = \frac{h}{|X|} \in (0, \frac{1}{10})$. We want to show that there exists some positive integer k satisfying $(1-x)^k \geq \frac{3}{4}$ and $(1-10x)^k \leq \frac{1}{4}$. Pick $k = \left\lceil \frac{\log \frac{1}{4}}{\log(1-10x)} \right\rceil$. Then $k \geq \frac{\log \frac{1}{4}}{\log(1-10x)} > 0$, which means that $(1-10x)^k \leq \frac{1}{4}$. We also have that $k \leq \frac{\log \frac{1}{4}}{\log(1-10x)} + 1 < \frac{\log \frac{3}{4}}{\log(1-x)}$ (since $x \in (0, \frac{1}{10})$), which means that $(1-x)^k \geq \frac{3}{4}$. □

B.5.2 Deterministic categorization

In this section we prove the following lemma (restated from Section B.3.2).

Lemma B.3.5. *There exists a deterministic categorization procedure which uses $O(n\sqrt{rh}\log r)$ queries over the whole run of the* Augmentation *algorithm.*

The main idea of the determinsitic categorization is the following: For each $v \in \bar{S} \setminus F_{\bar{S}}$, our deterministic categorization keeps track of a set $N_v \subseteq \mathsf{Ngh}_X(v)$ of h out-neighbors to v (if that many out-neighbors exist). Then we can either use N_v as a proof that v is heavy, or when we failed to find such a N_v we know that v is light.

In each phase, some of the vertices in N_v may be added to F (and thus removed from X). This may decrease the size of N_v. In this case we would like to find additional out-neighbors to add to N_v, until $|N_v| = h$, or determine that $|\mathsf{Ngh}_X(v)| < h$. One possible and immediate strategy would be to use Claim B.2.6 to find a new out-neighbor of v in $O(\log n)$ independence queries. However, adding arbitrary neighbors from $\mathsf{Ngh}_X(v) \setminus N_v$ will be expensive: over the whole run of the algorithm potentially every vertex in S will be added to N_v at some point which will require $\tilde{O}(nr)$ many independence queries in total for all N_v's—this is far too expensive than what we can allow. Instead, we want to be device a better strategy to pick $u \in \mathsf{Ngh}_X(v) \setminus N_v$.

Determinisitc strategy. For $u \in X$ we will denote by the *weight* of u, or $w(u)$, the number of sets N_v which contain u. Note that these weights change over the run of the algorithm. Also, note that the values $w(u)$ can be inferred from the sets N_v's which are known to the querier. Hence, we can assume that the querier knows the weights of elements in X. When u is moved from X to F, $w(u)$ new out-neighbors must be found, one for each $v \in \bar{S} \setminus F_{\bar{S}}$ for which the set N_v contained u.

This motivates the following strategy: *Whenever we need to find a new out-neighbors of v, we find $u \in \mathsf{Ngh}_X(v) \setminus N_v$ that minimizes $w(u)$.* To perform this strategy, we note that the binary-search idea from Claim B.2.6 can be implemented to find a u which minimizes $w(u)$. Indeed, if $\{u_1, u_2, \dots u_{|X|}\} \subseteq X$ with $w(u_1) \leq w(u_2) \leq \dots \leq w(u_{|X|})$, the binary search can first ask if there is an edge to $\{u_1, \dots, u_{\lfloor |X|/2 \rfloor}\}$ with a single query. If this was the case we recurse on $\{u_1, u_2, \dots u_{\lfloor |X|/2 \rfloor}\}$, otherwise recurse on $\{u_{\lfloor |X|/2 \rfloor + 1}, \dots, u_{|X|}\}$. This will guarantee that a the u_i which minimizes $w(u_i)$ will be found.[7]

Cost Analysis. For each $v \in \bar{S}$ we will at most once determine that N_v cannot be extended, i.e. that $|\mathsf{Ngh}_X(v)| < h$. This will require $O(n)$ independence queries in total. The remaining cost we will amortize over the vertices in $V = S \cup \bar{S}$. Consider that we find some out-neighbor $u \in X$ to some vertex $v \in \bar{S}$, using the above strategy. This uses $O(\log r)$ independence queries. We will charge this cost

[7] We actually use the same strategy to initialize the sets N_v: We discover out-neighbors u in the increasing order of $w(u)$.

to u if $w(u) \leq \frac{n\sqrt{h}}{\sqrt{r}}$, otherwise we will charge the cost to v. We make the following observations:

1. For $u \in S$, the total cost we charge to it at most $O(\frac{n\sqrt{h}}{\sqrt{r}} \log r)$.

2. For $v \in \bar{S}$, the total cost we charge to it is at most $O(\sqrt{rh} \log r)$.

Property 1 is easy to see, since we charge the cost $O(\log r)$ to it at most $O(\frac{n\sqrt{h}}{\sqrt{r}})$ times. To argue that Property 2 holds, let $u \in S$ be the first vertex which got added to N_v which had weight $w(u)$ strictly more than $\frac{n\sqrt{h}}{\sqrt{r}}$ (at the moment it was added to N_v). At this point in time, we know that for all remaining $u' \in \mathsf{Ngh}_X(v) \setminus N_v$, must have $w(u') \geq w(u) > \frac{n\sqrt{h}}{\sqrt{r}}$. Note that we can bound the total weight $\sum_{u \in X} w(u) = \sum_{v \in \bar{S} \setminus F_{\bar{S}}} |N_v| \leq nh$ at any point in time. Because of this upper bound, there can be at most $\frac{nh}{n\sqrt{h}/\sqrt{r}} = \sqrt{rh}$ such u'. Hence we can charge vertex v at most $\sqrt{rh}$ more times.

Since there are at most r vertices $u \in S$ and n vertices $v \in \bar{S}$, we conclude that the total cost (over all phases) for the deterministic categorization is $O(n\sqrt{rh} \log r)$. This proves Lemma B.3.5.

B.6 Open Problems

A major open problem is to close the big gap between upper and lower bounds for the matroid intersection problem with independent and rank queries. A major step towards this goal is to prove an $n^{1+\Omega(1)}$ lower bound. It will already be extremely interesting to prove such a bound for deterministic algorithms. It is also interesting to prove a cn lower bound for randomized algorithms for some constant $c > 1$ (the existing lower bound [Har08] holds only for deterministic algorithms).

Another major open problem is to understand whether the rank query is more powerful than the independence query. Are the tight bounds the same under both query-models? Two important intermediate steps towards answering this question is to achieve an $\tilde{O}(n\sqrt{r})$-query exact algorithm and an $\tilde{O}(n/\mathrm{poly}(\epsilon))$-query $(1-\epsilon)$-approximation algorithm under independence queries (such bounds have already been achieved under rank queries [CLSSW19]). We conjecture that the tight bounds are $\tilde{O}(n\sqrt{n})$ under both queries when $r = \Omega(n)$.

We believe that fully understanding the complexity of the reachability problem will be another major step towards understanding the matroid intersection problem. We conjecture that our $\tilde{O}(n\sqrt{n})$ bound is tight for $r = \Omega(n)$.

It is also very interesting to break the quadratic barrier for the weighted case. This barrier can be broken by a $(1-\epsilon)$-approximation algorithm by combining techniques from [CLSSW19; CQ16][8], but not the exact one.

[8] From a private communication with Kent Quanrud

Related problems are those for minimizing submodular functions. Proving an $n^{1+\Omega(1)}$ lower bound or subquadratic upper bound for, e.g., finding the minimizer of a submodular function or the non-trivial minimizer of a symmetric submodular function. Many recent studies (e.g. [RSW18; GPRW20; LLSZ20; MN19]) have led to some non-trivial bounds. However, it is still open whether an $n^{1+\Omega(1)}$ lower bound or an $n^{2-\Omega(1)}$ upper bound exist even in the special cases of computing minimum st-cut and hypergraph mincut in the cut query model.

Acknowledgment

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 715672. Jan van den Brand is partially supported by the Google PhD Fellowship Program. Danupon Nanongkai and Sagnik Mukhopadhyay are also partially supported by the Swedish Research Council (Reg. No. 2019-05622).

Bibliography

[AD71] Martin Aigner and Thomas A. Dowling. "Matching Theory for Combinatorial Geometries". In: *Transactions of the American Mathematical Society* 158.1 (1971), pp. 231–245 (cit. on pp. 193, 194, 198).

[CLSSW19] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. "Faster Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2019, pp. 1146–1168 (cit. on pp. 193–195, 198, 199, 201, 202, 209, 210, 215).

[CQ16] Chandra Chekuri and Kent Quanrud. "A Fast Approximation for Maximum Weight Matroid Intersection". In: *SODA*. SIAM, 2016, pp. 445–457 (cit. on p. 215).

[Cun86] William H. Cunningham. "Improved Bounds for Matroid Partition and Intersection Algorithms". In: *SIAM J. Comput.* 15.4 (1986), pp. 948–957 (cit. on pp. 193–195, 198, 200, 201).

[Edm70] Jack Edmonds. "Submodular functions, matroids, and certain polyhedra". In: *Combinatorial structures and their applications*. 1970, pp. 69–87 (cit. on p. 193).

[Edm79] Jack Edmonds. "Matroid intersection". In: *Annals of discrete Mathematics*. Vol. 4. Elsevier, 1979, pp. 39–49 (cit. on p. 193).

[EDVJ68] Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. "Matroid partition". In: *50 Years of Integer Programming 1958–2008* (1968), p. 199 (cit. on p. 193).

[GPRW20] Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S. Matthew Weinberg. "New Query Lower Bounds for Submodular Function Minimization". In: *ITCS*. Vol. 151. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 64:1–64:16 (cit. on p. 216).

[Har08] Nicholas J. A. Harvey. "Matroid intersection, pointer chasing, and Young's seminormal representation of S_n". In: *SODA*. SIAM, 2008, pp. 542–549 (cit. on p. 215).

[Law75] Eugene L. Lawler. "Matroid intersection algorithms". In: *Math. Program.* 9.1 (1975), pp. 31–56 (cit. on pp. 193, 194, 198).

[LLSZ20] Troy Lee, Tongyang Li, Miklos Santha, and Shengyu Zhang. "On the cut dimension of a graph". In: *CoRR* abs/2011.05085 (2020) (cit. on p. 216).

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. "A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization". In: *FOCS*. IEEE Computer Society, 2015, pp. 1049–1065 (cit. on p. 193).

[MN19] Sagnik Mukhopadhyay and Danupon Nanongkai. "Weighted Min-Cut: Sequential, Cut-Query and Streaming Algorithms". In: *CoRR* abs/1911.01651 (2019) (cit. on p. 216).

[Ngu19] Huy L. Nguyen. "A note on Cunningham's algorithm for matroid intersection". In: *CoRR* abs/1904.04129 (2019) (cit. on pp. 193–195, 198, 201, 202, 209).

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. "Computing Exact Minimum Cuts Without Knowing the Graph". In: *Proceedings of the 9th ITCS*. 2018, 39:1–39:16. DOI: 10.4230/LIPIcs.ITCS.2018.39 (cit. on p. 216).

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003 (cit. on p. 193).

Paper C

Breaking $O(nr)$ for Matroid Intersection

Joakim Blikstad

Article published in 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference). [Bli21]
Full version at `https://arxiv.org/abs/2105.05673`.

Abstract

We present algorithms that break the $\tilde{O}(nr)$-independence-query bound for the Matroid Intersection problem *for the full range of* r; where n is the size of the ground set and $r \leq n$ is the size of the largest common independent set. The $\tilde{O}(nr)$ bound was due to the efficient implementations [CLSSW FOCS'19; Nguy˜ên 2019] of the classic algorithm of Cunningham [SICOMP'86]. It was recently broken for large r ($r = \omega(\sqrt{n})$), first by the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\varepsilon)$-approximation algorithm of CLSSW [FOCS'19], and subsequently by the $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of BvdBMN [STOC'21]. No algorithm—even an approximation one—was known to break the $\tilde{O}(nr)$ bound for the full range of r. We present an $\tilde{O}(n\sqrt{r}/\varepsilon)$-query $(1-\varepsilon)$-approximation algorithm and an $\tilde{O}(nr^{3/4})$-query exact algorithm. Our algorithms improve the $\tilde{O}(nr)$ bound and also the bounds by CLSSW and BvdBMN for the full range of r.

C.1 Introduction

Matroid Intersection is a fundamental problem in combinatorial optimization that has been studied for more than half a century. The classic version of this problem is as follows: *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over a common ground set V of n elements, find the largest common independent set $S^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ by making independence oracle queries[1] of the form "Is $S \in \mathcal{I}_1$?" or "Is $S \in \mathcal{I}_2$?" for $S \subseteq V$.* The size of the largest common independent set is usually denoted by r.

Matroid intersection can be used to model many other combinatorial optimization problems, such as bipartite matching, arborescences, spanning tree packing, etc. As such, designing algorithms for matroid intersection is an interesting problem to study.

In this paper, we consider the task of finding a $(1-\varepsilon)$-approximate solution to the matroid intersection problem, that is finding some common independent set S of size at least $(1-\varepsilon)r$. We show an improvement of approximation algorithms for matroid intersection, and as a consequence also obtain an improvement for the *exact* matroid intersection problem.

Previous work. Polynomial algorithms for matroid intersection started with the work of Edmond's $O(n^2 r)$-query algorithms [EDVJ68; Edm70; Edm79] in the 1960s. Since then, there has been a long line of research e.g. [AD71; Law75; Cun86; LSW15; CQ16; CLSSW19; BBMN21]. Cunningham [Cun86] designed a $O(nr^{1.5})$-query blocking-flow algorithm in 1986, similar to that of Hopcroft-Karp's bipartite-matching or Dinic's maximum-flow algorithms. Chekuri and Quanrud [CQ16] pointed out that Cunningham's classic algorithm [Cun86] from 1986 is already a $O(nr/\varepsilon)$-query $(1-\epsilon)$-approximation algorithm. Recently, Chakrabarty-Lee-Sidford-Singla-Wong [CLSSW19] and Nguy˜ên [Ngu19] independently showed how to implement Cunningham's classic algorithm using only $\tilde{O}(nr)$ independence queries. This is akin to spending $\tilde{O}(n)$ queries to find each of the so-called *augmenting paths*. A fundamental question is whether several augmenting paths can be found simultaneously to break the $\tilde{O}(nr)$ bound.

This question has been answered for large r ($r = \omega(\sqrt{n})$), first by the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\epsilon)$-approximation algorithm of Chakrabarty-Lee-Sidford-Singla-Wong[2] [CLSSW19], and very recently by the randomized $\tilde{O}(n^{6/5} r^{3/5})$-query exact algorithm of Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai [BBMN21]. Whether we can break the $O(nr)$-query bound *for the full range of r* remained open *even for approximation algorithms*.

[1] There are also other oracle models considered in the literature (e.g. rank-oracles), but in this paper we focus on the independence query model. Whenever we say *query* in this paper, we thus mean *independence query*.

[2] In the same paper they also show a $\tilde{O}(n^2 r^{-1} \varepsilon^{-2} + r^{1.5} \varepsilon^{-4.5})$-query algorithm.

Our results. We break the $O(nr)$-query bound for both *approximation* and *exact* algorithms. We first state our results for approximate matroid intersection.[3]

Theorem C.1.1 (Approximation algorithm)**.** *There is a deterministic algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (1-\varepsilon)r$, using $O\left(\frac{n\sqrt{r \log r}}{\varepsilon}\right)$ independence queries.*

Plugging Theorem C.1.1 in the framework of [BBMN21], we get an improved algorithm—more efficient than the previous state-of-the-art—for exact matroid intersection which we state next.

Theorem C.1.2 (Exact algorithm)**.** *There is a randomized algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum cardinality r, and w.h.p.[4] uses $O(nr^{3/4} \log n)$ independence queries. There is also a deterministic exact algorithm using $O(nr^{5/6} \log n)$ queries.*

Remark C.1.3. Although we only focus on the query-complexity in this paper, we note that the time-complexity of the algorithms are dominated by query-oracle calls. That is, our approximation algorithm runs in $\tilde{O}(n\sqrt{r}\mathcal{T}_{\text{ind}}/\varepsilon)$ time, and the exact algorithms in $\tilde{O}(nr^{3/4}\mathcal{T}_{\text{ind}})$ (randomized) respectively $\tilde{O}(nr^{5/6}\mathcal{T}_{\text{ind}})$ time (deterministic), where $\mathcal{T}_{\text{ind}}$ denotes the time-complexity of the independence-oracle.

C.1.1 Technical Overview

Approximation algorithm. Our approximation algorithm (Theorem C.1.1) is a modified version of Chakrabarty-Lee-Sidford-Singla-Wong's $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query approximation algorithm [CLSSW19, Section 6]. The algorithm is based on the ideas of Cunningham's classic blocking-flow algorithm [Cun86] and runs in $O(1/\varepsilon)$ phases, where in each phase the algorithm seeks to find a *maximal* set of *augmentations* in the *exchange graph*. Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph (with bipartition $(S + \{s,t\}, V \setminus S)$). Finding a shortest (s,t)-path, called an *augmenting path*, in $G(S)$ means one can increase the size of the common independent set S by 1. Since the exchange graph changes after each augmentation,[5] and we do not know how to find a single augmenting path

[3] The $\tilde{O}(n^2 r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm of [CLSSW19] is the only previous algorithm which is more efficient than our algorithm in some range of r and ε. Actually, since the $\tilde{O}(n^2 r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm use the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ algorithm as a subroutine, we do get a slightly improved version by using our $\tilde{O}(n\sqrt{r}/\varepsilon)$ algorithm as the subroutine instead: $\tilde{O}(n^2 r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4})$.

[4] *w.h.p. = with high probability* meaning with probability $1 - n^{-c}$ for some arbitrarily large constant c.

[5] Unlike what happens in augmenting path algorithms for flow and bipartite matching, where the underlying graphs remain the same.

faster than $\Omega(n)$ queries, the need to find several augmentations in parallel arises. [CLSSW19, Section 6] introduces the notion of *augmenting sets*: a generalization of the classical *augmenting paths* but where one can perform many augmentations in parallel.

So the revised goal of the algorithm is to, in each phase, efficiently find a *maximal augmenting set* (akin to a *blocking-flow* in bipartite matching or flow algorithms). Towards this goal, the algorithm maintains a relaxed version of augmenting set—called a *partial* augmenting set—and keeps *refining* it to make it "better" (i.e. closer to a maximal augmenting set). Here we give two independent improvements on top of the algorithm of [CLSSW19]:

1. The algorithm of [CLSSW19] refines the partial augmenting set by a sequence of operations on two adjacent distance layers in the exchange graph. In our algorithm, we instead consider *three* consecutive layers for our basic refinement procedures. This lets us focus our analysis on what happens in S—the "left" side of the bipartite exchange graph—which contains at most r elements in total (in contrast to [CLSSW19] where the performance analysis is dependent on all n elements). The number of times we need to run the refinement procedures thus depends on r, instead of n, which makes the algorithm faster when $r = o(n)$.

2. When the partial augmenting set is "close enough" to a maximal augmenting set, [CLSSW19] falls back to finding the remaining augmenting paths one at a time. In our algorithm, we also change to a different procedure when the partial augmenting set is close enough to maximal. The difference is that, instead of finding arbitrary augmenting paths, we find a special type of *valid paths* with respect to the partial augmenting set, so that these paths can be used to further improve (refine) the partial augmenting set. The number of valid paths we need to find is less than the number of augmenting paths [CLSSW19] needs to find. This decreases the dependency on ε in the final algorithm.

The first improvement (Item 1) replaces the $\sqrt{n}$ term with a $\sqrt{r}$ term in the query complexity of the algorithm. The second improvement (Item 2) shaves off a $1/\sqrt{\varepsilon}$ term from the query complexity. Together they thus bring down the query complexity from $\tilde{O}(\frac{n\sqrt{n}}{\varepsilon\sqrt{\varepsilon}})$ in [CLSSW19] to $\tilde{O}(\frac{n\sqrt{r}}{\varepsilon})$ as in our Theorem C.1.1. Note that these two improvements are independent of each other, and can be applied individually.

Exact algorithm. To obtain the exact algorithm (Theorem C.1.2), we use the framework of Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai's $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm [BBMN21]. The main idea of this algorithm is to combine approximation algorithms—which can efficiently find a common independent set only εr away from the optimal—with a randomized $\tilde{O}(n\sqrt{r})$-query subroutine to find each of the remaining *few, very long* augmenting paths. The $\tilde{O}(n^{6/5}r^{3/5})$-query

exact algorithm [BBMN21] currently uses Chakrabarty-Lee-Sidford-Singla-Wong's $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ approximation algorithm [CLSSW19] as a subroutine. Simply replacing it with our improved approximation algorithm (Theorem C.1.1) yields our $\tilde{O}(nr^{3/4})$-query exact algorithm.

C.2 Preliminaries

We use the standard definitions of *matroid* $\mathcal{M} = (V, \mathcal{I})$; *rank* $\text{rank}(X)$ for any $X \subseteq V$; *exchange graph* $G(S)$ for a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$; and *augmenting paths* in $G(S)$ throughout this paper. For completeness, we define them below. We also need the notions of *augmenting sets* introduced by [CLSSW19], which we also define in later this section.

Matroids

Definition C.2.1 (Matroid)**.** A *matroid* is a tuple $\mathcal{M} = (V, \mathcal{I})$ of a *ground set* V of n elements, and non-empty family $\mathcal{I} \subseteq 2^V$ of *independent sets* satisfying

Downward closure: if $S \in \mathcal{I}$, then $S' \in \mathcal{I}$ for all $S' \subseteq S$.

Exchange property: if $S, S' \in \mathcal{I}$, $|S| > |S'|$, then there exists $x \in S \setminus S'$ such that $S' \cup \{x\} \in \mathcal{I}$.

Definition C.2.2 (Set notation)**.** We will use $A + x$ and $A - x$ to denote $A \cup \{x\}$ respectively $A \setminus \{x\}$, as is usual in matroid intersection literature. We will also use $\bar{A} := V \setminus A$, $A + B := A \cup B$, and $A - B := A \setminus B$.

Definition C.2.3 (Matroid rank)**.** The *rank* of $A \subseteq V$, denoted by $\text{rank}(A)$, is the size of the largest (or, equivalently, any maximal) independent set contained in A. It is well-known that the rank-function is submodular, i.e. $\text{rank}(A + x) - \text{rank}(A) \geq \text{rank}(B + x) - \text{rank}(B)$ whenever $A \subseteq B \subseteq V$ and $x \in V \setminus B$.[6] Note that $\text{rank}(A) = |A|$ if and only if $A \subseteq \mathcal{I}$.

Definition C.2.4 (Matroid Intersection)**.** Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set V, a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. The *matroid intersection problem* asks us to find the largest common independent set—whose cardinality we denote by r. We use rank_1 and rank_2 to be the rank functions of the corresponding matroids.

The Exchange Graph

Many matroid intersection algorithms, e.g. those in [Edm79; AD71; Law75; Cun86; Ngu19; BBMN21], are based on iteratively finding *augmenting paths* in the *exchange graph*.

[6] Usually denoted as the *diminishing returns* property of submodular functions.

Definition C.2.5 (Exchange graph)**.** Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set, and a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph on vertex set $V \cup \{s, t\}$ with the following arcs (or directed edges):

1. (s, b) for $b \in \bar{S}$ when $S + b \in \mathcal{I}_1$.
2. (b, t) for $b \in \bar{S}$ when $S + b \in \mathcal{I}_2$.
3. (a, b) for $b \in \bar{S}, a \in S$ when $S + b - a \in \mathcal{I}_1$.
4. (b, a) for $b \in \bar{S}, a \in S$ when $S + b - a \in \mathcal{I}_2$.

We will denote the set of elements at distance k from s by the distance-layer D_k.

Definition C.2.6 (Shortest augmenting path)**.** In $G(S)$, a shortest (s, t)-path $p = (s, b_1, a_1, b_2, a_2, \ldots, a_\ell, b_{\ell+1}, t)$ (with $b_i \in \bar{S}$ and $a_i \in S$) is called a *shortest augmenting path*. We can *augment* S along the path p to obtain $S' = S \oplus p = S + b_1 - a_1 + b_2 - a_2 \ldots + b_{\ell+1}$, which is well-known to also be a common independent set (with $|S'| = |S| + 1$) [Cun86]. Conversely, there must exist a shortest augmenting path whenever $|S| < r$.

The following lemma is very useful for $(1 - \varepsilon)$-approximation algorithms since it essentially says that one needs only to consider paths up to length $O(\frac{1}{\varepsilon})$.

Lemma C.2.7 (Cunningham [Cun86])**.** *If the length of the shortest (s, t)-path in $G(S)$ is at least $2\ell + 2$, then $|S| \geq (1 - O(1/\ell))r$.*

Lemma C.2.8 (Exchange discovery by binary search [CLSSW19; Ngu19])**.** *Suppose $\mathcal{M} = (V, \mathcal{I})$ is a matroid, $Y \subseteq X \in \mathcal{I}$, and $b \notin X$ such that $X + b \notin \mathcal{I}$. Then, using $O(\log |Y|)$ independence queries one can find some $a \in Y$ such that $X + b - a \in \mathcal{I}$ or determine that none exist.*[7]

Augmenting Sets

A generalization of the classical *augmenting paths*—called *augmenting sets*—play a key role in the approximation algorithm of [CLSSW19], and therefore also in the modified version of this algorithm presented in this paper. In order to efficiently find "good" augmenting sets, the algorithm works with a relaxed form of them instead: *partial* augmenting sets. The following definitions and key properties of (partial) augmenting sets are copied from [CLSSW19] where one can find the corresponding proofs.

[7] When $X = S$, we can use this to find edges of type 3 and 4 in the exchange graph.

Definition C.2.9 (Augmenting Sets, from [CLSSW19, Definition 24])**.** Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $G(S)$ be the corresponding exchange graph with shortest (s,t)-path of length $2(\ell+1)$ and distance layers $D_1, D_2, \ldots, D_{2\ell+1}$. A collection of sets $\Pi_\ell := (B_1, A_1, B_2, A_2, \ldots, A_\ell, B_{\ell+1})$ form an *augmenting set* (of *width* w) in $G(S)$ if the following conditions are satisfied:

(a) For $1 \leq k \leq \ell+1$, we have $A_k \subseteq D_{2k}$ and $B_k \subseteq D_{2k-1}$.

(b) $|B_1| = |A_1| = |B_2| = \cdots = |B_{\ell+1}| = w$

(c) $S + B_1 \in \mathcal{I}_1$

(d) $S + B_{\ell+1} \in \mathcal{I}_2$

(e) For all $1 \leq k \leq \ell$, we have $S - A_k + B_{k+1} \in \mathcal{I}_1$

(f) For all $1 \leq k \leq \ell$, we have $S - A_k + B_k \in \mathcal{I}_2$

Definition C.2.10 (Partial Augmenting Sets, from [CLSSW19, Definition 37])**.** We say that $\Phi_\ell := (B_1, A_1, B_2, A_2, \ldots, A_\ell, B_{\ell+1})$ forms a *partial augmenting set* if it satisfies the conditions (a), (c), (d), and (e) of an *augmenting set*, plus the following two relaxed conditions:

(b) $|B_1| \geq |A_1| \geq |B_2| \geq \cdots \geq |B_{\ell+1}|$.

(f) For all $1 \leq k \leq \ell$, we have $\operatorname{rank}_2(S - A_k + B_k) = \operatorname{rank}_2(S)$.

Theorem C.2.11 (from [CLSSW19, Theorem 25])**.** *Let* $\Pi_\ell := (B_1, A_1, B_2, A_2, \ldots, B_\ell, A_\ell, B_{\ell+1})$ *be the an augmenting set in the exchange graph* $G(S)$*. Then the set* $S' := S \oplus \Pi_\ell := S + B_1 - A_1 + B_2 - \cdots + B_\ell - A_\ell + B_{\ell+1}$ *is a common independent set.*[8]

We also need the notion of *maximal* augmenting sets, which naturally correspond to a maximal ordered collection of shortest augmenting paths, where, after augmentation, the (s,t)-distance must have increased. The following are due to [CLSSW19].

Definition C.2.12 (Maximal Augmenting Sets, from [CLSSW19, Definition 35])**.** Let $\Pi_\ell = (B_1, A_1, B_2, \cdots, B_\ell, A_\ell, B_{\ell+1})$ and $\tilde{\Pi}_\ell = (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2, \cdots, \tilde{B}_\ell, \tilde{A}_\ell, \tilde{B}_{\ell+1})$ be two augmenting sets in $G(S)$. We say $\tilde{\Pi}_\ell$ *contains* Π_ℓ if $B_k \subseteq \tilde{B}_k$ and $A_k \subseteq \tilde{A}_k$, for all k. An augmenting set Π_ℓ is called *maximal* if there exists no other augmenting set $\tilde{\Pi}_\ell$ containing Π_ℓ.

Theorem C.2.13 (from [CLSSW19, Theorem 36])**.** *An augmenting set* Π_ℓ *is maximal if and only if there is no augmenting path of length at most* $2(\ell+1)$ *in* $G(S \oplus \Pi_\ell)$.

[8] Note that $|S'| = |S| + w$, where w is the width of Π_ℓ. In particular, an augmenting set with width $w = 1$ is exactly an augmenting path.

C.3 Improved Approximation Algorithm

Our algorithm closely follows the algorithm of Chakrabarty-Lee-Sidford-Singla-Wong [CLSSW19, Section 6]. The algorithm runs in phases, where in each phase the algorithm finds a maximal set of augmentations to perform, so that the (s,t)-distance in the exchange graph increases between phases. By Lemma C.2.7, only $O(1/\varepsilon)$ phases are necessary.

In the beginning of a phase, the algorithm runs a breadth-first-search to compute the distance layers $D_1, D_2, \dots D_{2\ell+1}$ in the exchange graph $G(S)$, where S is the current common independent set. The total number of independence queries, across all phases, for these BFS's can be bounded by $O(n \log(r)/\varepsilon)$. We refer to [CLSSW19, Algorithm 4, Lemma 19, and Proof of Theorem 21] for how to implement such a BFS efficiently.

After the distance layers have been found, the search for a maximal augmenting set begins. We start by summarizing on a high level how the algorithm of [CLSSW19] does this in two stages:

1. The first stage keeps track of a *partial* augmenting set which it keeps *refining* by a series of operations on adjacent distance layers in the exchange graph, to make it closer to a *maximal* augmenting set.

2. When we are "close enough" to a *maximum* augmenting set, the second stage handles the last few augmenting paths—for which the first stage slows down—by finding the remaining augmenting paths individually one at a time.

Here we give two independent improvements over the algorithm of [CLSSW19], one for each stage. The first improvement is to replace the refine operations in the first stage by a new subroutine `RefineABA` (Line 4) working on *three* consecutive layers instead of two. This allows us to measure progress in terms of r instead of n. The second improvement is for the second stage where we, instead of finding arbitrary augmenting paths, work directly on top of the output of the first stage and find a specific type of *valid paths* with respect to the partial augmenting set, using a new a subroutine `RefinePath` (Section C.3.2).

C.3.1 Implementing a Phase: Refining

The basic refining ideas and procedures in this section are the same as in [CLSSW19]. The goal is to keep track of a partial augmenting set $\Phi_\ell = (B_1, A_1, B_2, \dots, A_\ell, B_{\ell+1})$ which is iteratively made "better" through some *refine procedures*. Eventually, the partial augmenting set will become a maximal augmenting set, which concludes the phase. Towards this goal, we maintain three types of elements in each layer:

Selected. Denoted by A_k or B_k. These form the partial augmenting set $\Phi_\ell = (B_1, A_1, B_2, \dots, A_\ell, B_{\ell+1})$.

Removed. Denoted by R_k. These elements are safe to disregard from further computation (i.e. deemed useless) when refining Φ_ℓ towards a maximal augmenting set.

Fresh. Denoted by F_k. These are the elements that are neither *selected* nor *removed*.

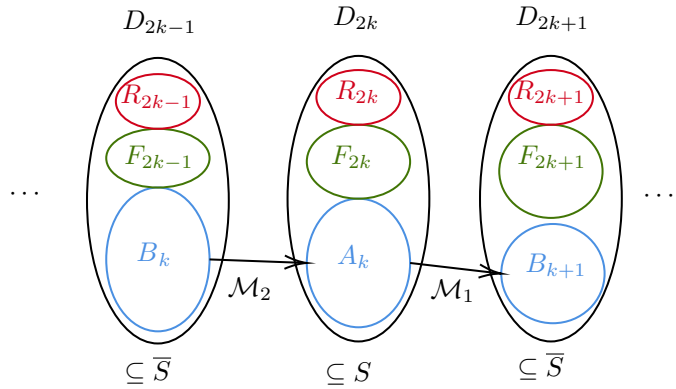


Figure C.1: An illustration of a few neighboring layers. Note that $(B_k, R_{2k-1}, F_{2k-1})$ form a partition of odd layer $D_{2k-1} \subseteq \bar{S}$, and (A_k, R_{2k}, F_{2k}) form a partition of even layer $D_{2k} \subseteq S$.

Elements can change their types from *fresh* $\to$ *selected* $\to$ *removed*, but never in the other direction. Initially, we start with all elements being fresh.[9] For convenience, we also define "imaginary" layers D_0 and $D_{2\ell+2}$ with $A_0 = R_0 = F_0 = D_0 = A_{\ell+1} = R_{2\ell+2} = F_{2\ell+2} = D_{2\ell+2} = \emptyset$. The algorithm maintains the following *phase invariants* (which are initially satisfied) during the refinement process:

Definition C.3.1 (Phase Invariants, from [CLSSW19, Section 6.3.2])**.** The *phase invariants* are:

(a-b) $\Phi_\ell = (B_1, A_1, B_2, \ldots, A_\ell, B_{\ell+1})$ forms a partial augmenting set.[10]

(c) For $1 \le k \le \ell$, for any $X \subseteq B_{k+1} + F_{2k+1} = D_{2k+1} - R_{2k+1}$, if $S - (A_k + R_{2k}) + X \in \mathcal{I}_1$ then $S - A_k + X \in \mathcal{I}_1$. [11]

(d) $\operatorname{rank}_2(W + R_{2k-1}) = \operatorname{rank}_2(W)$ where $W = S - (D_{2k} - R_{2k}) + B_k$.

[9] This differs slightly from [CLSSW19], where the initially B_1 is greedily picked to be maximal so that $S + B_1 \in \mathcal{I}_1$, while the rest of the elements are fresh.

[10] The naming of this invariant as (a-b) is to be consistent with [CLSSW19] where this condition is split up into two separate items (a) and (b).

[11] An equivalent condition for (c) is: $\operatorname{rank}_1(W - R_{2k}) = \operatorname{rank}_1(W) - |R_{2k}|$, where $W = S - A_k + (D_{2k+1} - R_{2k+1})$.

Remark C.3.2. Invariant (c) essentially says that if R_{2k+1} is "useless", then so is R_{2k}. Similarly, Invariant (d) says that if R_{2k} is "useless", then so is R_{2k-1}. Together they imply that we can safely ignore all the removed elements.

Lemma C.3.3. *Suppose that (i) the phase invariants hold; (ii)* $|B_1| = |A_1| = \cdots = |B_{\ell+1}|$*; and (iii)* B_1 *is a maximal subset of* $D_1 \setminus R_1$ *satisfying* $S + B_1 \in \mathcal{I}_1$*. Then* $(B_1, A_1, \ldots, B_{\ell+1})$ *is a maximal augmenting set.*

Proof idea. (See [CLSSW19, Proof of Lemma 44] for a complete proof). If it was not maximal, there exists an augmenting path $(b_1, a_1, \ldots, b_{\ell+1})$ in the exchange graph after augmenting along $(B_1, A_1, \ldots, B_{\ell+1})$. However, (iii) then says that b_1 must have been removed since it cannot be fresh. But if b_1 is removed, then so was a_1, then so was b_2 etc., by invariants (c) and (d) (this requires a technical, but straightforward, argument). However, $b_{\ell+1}$ cannot have been removed (by invariant (d)), which gives the desired contradiction. □

Refining Two Adjacent Layers

We now present the basic refinement procedures from [CLSSW19], which are operations on neighboring layers. There is some asymmetry in how (odd, even) and (even, odd) layer-pairs are handled, arising from the inherent asymmetry of the independence query between S and $\bar{S}$, but the ideas are the same.

`RefineAB`(k) extends B_{k+1} as much as possible while respecting invariant (a-b) (Lines 1-2). Then a maximal collection of element in A_k which can be "matched" to B_{k+1} is found, and the others elements in A_k are removed (Lines 3-4).

`RefineBA`(k) finds a maximal subset B_k that can be "matched" to $A_k + F_{2k}$, and removes the other elements of B_k (Lines 1-2). Then A_k is extended with elements from F_{2k} which are the endpoints of the above "matching" (Lines 3-4).

Algorithm C.1: `RefineAB`(k) (called `Refine1` in [CLSSW19, Algorithm 9])

1 Find maximal $B \subseteq F_{2k+1}$ s.t. $S - A_k + B_{k+1} + B \in \mathcal{I}_1$
2 $B_{k+1} \longleftarrow B_{k+1} + B, F_{2k+1} \longleftarrow F_{2k+1} - B$
3 Find maximal $A \subseteq A_k$ s.t. $S - A_k + B_{k+1} + A \in \mathcal{I}_1$
4 $A_k \longleftarrow A_k - A, R_{2k} \longleftarrow R_{2k} + A$

The following properties of the `RefineAB` and `RefineBA` methods are proven in [CLSSW19].

Lemma C.3.4 (from [CLSSW19, Lemmas 40-42])**.** *Both* ***`RefineAB`*** *and* ***`RefineBA`*** *preserve the invariants. Also: after* ***`RefineAB`***(k) *is run, we have* $|A_k| = |B_{k+1}|$ *(unless* $k = 0$*). After* ***`RefineBA`***(k) *is run, we have* $|B_k| = |A_k|$ *(unless* $k = \ell + 1$*).*

Algorithm C.2: `RefineBA`(k) (called `Refine2` in [CLSSW19, Algorithm 10])

1 Find maximal $B \subseteq B_k$ s.t. $S - (D_{2k} - R_{2k}) + B \in \mathcal{I}_2$
2 $R_{2k-1} \longleftarrow R_{2k-1} + B_k \backslash B, B_k \longleftarrow B$
3 Find maximal $A \subseteq F_{2k}$ s.t. $S - (D_{2k} - R_{2k}) + B_k + A \in \mathcal{I}_2$
4 $A_k \longleftarrow A_k + F_{2k} \backslash A, F_{2k} \longleftarrow A$

Lemma C.3.5 (from [CLSSW19, Lemma 45])**.** *`RefineAB` can be implemented with $O(|D_{2k}| + |D_{2k+1}|)$ queries. `RefineBA` can be implemented with $O(|D_{2k-1}| + |D_{2k}|)$ queries.*

Observation C.3.6. *Lemma C.3.4 is particularly interesting. It says that at least $|A_k^{old}| - |B_{k+1}^{old}|$ (respectively $|B_k^{old}| - |A_k^{old}|$) elements change type when running `RefineAB` (respectively `RefineAB`).*

Remark C.3.7. Observation C.3.6 is used in [CLSSW19] to bound the number of times one needs to refine the partial augmenting set. Indeed, every element can only change its type a constant number of times. In a single refinement pass, procedures `RefineAB`(k) and `RefineBA`(k) are called for all k, and we obtain a telescoping sum guaranteeing us that $|B_1^{old}| - |B_{\ell+1}^{old}|$ elements have changed their types. Hence, after $O(\sqrt{n})$ refinement passes we have $|B_1| - |B_{\ell+1}| \leq \sqrt{n}$, and we are "close" to having a maximal augmenting set—only around $\sqrt{n}$ many augmenting paths away. This is essentially what lets [CLSSW19] obtain their subquadratic $\tilde{O}(n^{1.5}/\text{poly}(\varepsilon))$ algorithm.

Refining Three Adjacent Layers

We are now ready to present the new `RefineABA` method (Algorithm C.3), which is **not** present in [CLSSW19]. This method works similarly to `RefineAB` and `RefineBA`, but on **three** (instead of two) consecutive layers $(D_{2k}, D_{2k+1}, D_{2k+2})$ with the corresponding sets (A_k, B_{k+1}, A_{k+1}).

The motivation for this new procedure is that we can get a stronger version of Observation C.3.6: after running `RefineABA`(k) we want that at least $|A_k^{old}| - |A_{k+1}^{old}|$ element in **even** layers have changed types. Note that there are at most $|S| \leq r$ elements in the even layers (as opposed to n elements in total, which can be much larger), so this means we need to refine the partial augmenting set fewer times when using `RefineABA` compared to when just using `RefineAB` and `RefineBA`. In particular, we will get that after $O(\sqrt{r})$ refinement passes, $|B_1| - |B_{\ell+1}| \leq \sqrt{r}$.

Remark C.3.8. A natural question to ask is if it actually could be the case that only elements in odd layers (i.e. those in $\bar{S}$ which there are up to n many of) change their type (while elements in even layers do not) during the refinement passes in the algorithm of [CLSSW19] (which only uses the two-layer refinement procedures)? That is, is the new three-layer refinement procedure necessary? The

answer is yes. Consider for example the case with 5 layers $B_1 \subseteq D_1; A_1 \subseteq D_2; B_2 \subseteq D_3; A_2 \subseteq D_4; B_3 \subseteq D_5$ where $q := |B_1| = |A_1|$ and $|A_2| = |B_3| = 0$. Refining the consecutive pair (B_1, A_1) or (A_2, B_3) will not do anything. When refining (A_1, B_2) it could be the case that only B_2 increases (say any q-size subset in D_3 can be "matched" with A_1). Similarly, when refining (B_2, A_2) it could be the case that only B_2 decreases (say there is only a single element in D_3 which could be "matched" with anything in the next layer D_4, then it is unlikely that this specific element is already selected in B_2). In this case, we would need to run the two-layer refinement procedures around $|D_3|/q \approx n/q$ times before anything other than B_2 changes. In contrast, the new `RefineABA` method would, when run on (A_1, B_2, A_2), terminate with $|A_1| = |B_2| = |A_2|$ (that is it would have found the "special" element in D_3 the first time it is run).

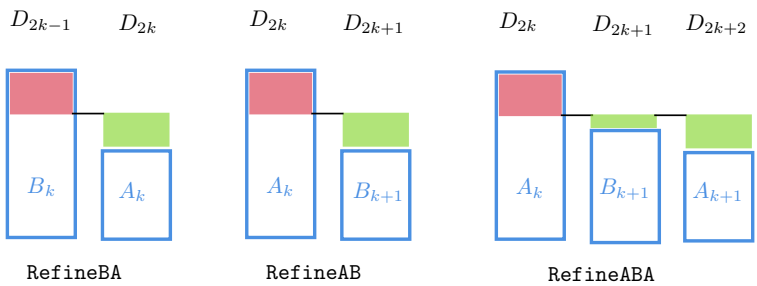


Figure C.2: An illustration how the different refine methods change the partial augmenting sets. Newly selected elements are marked in green, while newly removed elements are marked in red.

To explain how `RefineABA` works, let us start with a simple case, namely when $S = \emptyset$, i.e. there is only one layer between s and t in the exchange graph. Here, finding a maximal augmenting set is the same as finding some maximal set B which is independent in both matroids. Running `RefineAB` would extend this B with elements as long as it is independent in the first matroid (ignoring the second matroid), while `RefineBA` would throw away elements from B until it is independent in the second matroid (now ignoring the first matroid). If we just alternate running `RefineAB` and `RefineBA` we would in the worst case need to do this up to n times (which is too expensive). Instead, there is a very simple greedy algorithm that efficiently finds a maximal set B independent in both of the matroids[12]: *for each element, include it in B if this does not break independence for either matroid.* This is akin to how our `RefineABA` method works: it looks at the constraints from both matroids simultaneously (both neighboring layers) and greedily selects B.

In the general case, `RefineABA` can be seen as running `RefineAB` and `RefineBA` simultaneously. The algorithm starts by asserting $|B_{k+1}| = |A_{k+1}|$ (so that $S + B_{k+1} - A_{k+1} \in \mathcal{I}_2$) by running `RefineBA`. So now we have both $S + B_{k+1} - A_k \in \mathcal{I}_1$

[12]This algorithm on its own is a well-known $\frac{1}{2}$-approximation algorithm for matroid intersection.

and $S+B_{k+1}-A_{k+1} \in \mathcal{I}_2$, and the algorithm proceeds to greedily extend B_{k+1} while it is still consistent with both the previous layer A_k and the next layer $A_{k+1}+F_{2k+2}$. Some care has to be taken here to also mark elements as removed to preserve the phase invariants. Finally, the algorithm decreases the size of A_k, respectively increases the size of A_{k+1}, to both match $|B_{k+1}|$.

Algorithm C.3: `RefineABA`(k)

```
1  RefineBA(k + 1)
2  for x ∈ F_{2k+1} do
3      if S − A_k + B_{k+1} + x ∈ I_1 then
4          if S − A_{k+1} − F_{2k+2} + B_{k+1} + x ∈ I_2 then
               // Select x
5              B_{k+1} ← B_{k+1} + x
6              F_{2k+1} ← F_{2k+1} − x
7          else
               // Remove x
8              R_{2k+1} ← R_{2k+1} + x
9              F_{2k+1} ← F_{2k+1} − x
10 RefineBA(k + 1)
11 RefineAB(k)
```

We now state some properties of `RefineABA`. These properties are relatively straightforward—although technical and notation-heavy—to prove.

Lemma C.3.9. *`RefineABA`(k) preserves the phase invariants.*

Lemma C.3.10. *After `RefineABA`(k) is run, we have $|A_k| = |B_{k+1}| = |A_{k+1}|$ (unless $k = 0$ or $k = \ell$, where the sets $A_0 = A_{\ell+1} = \emptyset$ are "imaginiary").*

Lemma C.3.11. *`RefineABA`(k) uses $O(|D_{2k}| + |D_{2k+1}| + |D_{2k+2}|)$ independence queries.*

Proof of Lemma C.3.9. Intuitively, the only tricky part is showing that invariant (c) is preserved when some x is removed in line 7. We can pretend that we add x to B_{k+1} temporarily, and then run `RefineBA`$(k+1)$ in a way which would remove this x immediately (and thus removing x did indeed preserve the invariants). We present a formal proof below.

We already know that `RefineAB` and `RefineBA` preserve the invariants by Lemma C.3.4, so it suffices to check that the for-loop starting in line 2 preserves the invariants. We verify that this is the case after processing each $x \in F_{2k+1}$ in the for-loop:

Invariant (a-b) holds by design: when x is added to B_{k+1} we know both that $S - A_k + B_{k+1} + x \in \mathcal{I}_1$ and $\text{rank}_2(S - A_{k+1} + B_{k+1})$ cannot decrease. Note also that $\text{rank}_2(S - A_{k+1} + B_{k+1}) \leq \text{rank}_2(S)$ when $k + 1 \leq \ell$ too (so it cannot increase either), since otherwise there must exist some $b \in B_{k+1}$ so that $S + b \in \mathcal{I}_2$ (by the matroid exchange property) which is impossible since we are not in the last layer (the layer preceding t in $G(S)$).

Invariant (c) trivially holds, since the set $B_{k+1} + F_{2k+1}$ will only decrease, which only restricts the choice of $X \subseteq B_{k+1} + F_{2k+1}$.

Invariant (d) will also be preserved. We need to argue that this is the case when x is removed in line 7. Let $W := S - A_{k+1} - F_{2k+2} + B_{k+1} = S - (D_{2k+2} - R_{2k+2}) + B_{k+1}$, and R^{old}_{2k+1} be the set R_{2k+1} before x was added to it. First note that $W \in \mathcal{I}_2$, since this holds after the `RefineBA` call in line 1, (since $|A_{k+1}| = |B_{k+1}|$ after this call) and B_{k+1} is only extended with elements which preserve this property. This means that $\text{rank}_2(W + x) = \text{rank}_2(W) = |W|$, since $W + x = S - A_{k+1} - F_{2k+2} + B_{k+1} + x \notin \mathcal{I}_2$. Since the invariant held before, we also know that $\text{rank}_2(W + R^{old}_{2k+1}) = \text{rank}_2(W) = |W|$. Hence W is a maximal independent (in $\mathcal{M}_2$) subset of $W + R^{old}_{2k+1} + x$, as neither x nor elements from R^{old}_{2k+1} can be used to extend it. Hence $\text{rank}_2(W + R^{old}_{2k+1} + x) = |W| = \text{rank}_2(W)$; that is invariant (d) is preserved. □

Proof of Lemma C.3.10. We focus our attention on the `RefineBA` and `RefineAB` calls in lines 8-9, and argue that they do not change B_{k+1}. This would prove the lemma, since by Lemma C.3.4 we would then have $|A_k| = |B_{k+1}|$ and $|B_{k+1}| = |A_{k+1}|$.

Indeed, `RefineBA`$(k+1)$ finds a maximal $B \subseteq B_{k+1}$ such that $S - (D_{2k+2} - R_{2k+2}) + B \subseteq \mathcal{I}_2$, and remove all elements not in B from B_{k+1}. Here, $B = B_{k+1}$ will be found, since $S - (D_{2k+2} - R_{2k+2}) + B_{k+1} \in \mathcal{I}_2$ after the for-loop in line 2 of `RefineABA`.

Similarly, we see that `RefineAB`(k) finds a maximal $B \subseteq F_{2k+1}$ such that $S - A_k + B_{k+1} + B \in \mathcal{I}_1$, and extend B_{k+1} with this B. However, only $B = \emptyset$ works, since each $x \in F_{2k+1}$ for which $S - A_k + B_{k+1} + x \in \mathcal{I}_1$ was either selected or removed in lines 5 or 7. □

Proof of Lemma C.3.11. It is easy to see that `RefineAB`(k) uses $O(|D_{2k}| + |D_{2k+1}|)$ queries, and `RefineBA`$(k+1)$ uses $O(|D_{2k+1}| + |D_{2k+2}|)$ queries. The for-loop in line 2 will use $O(|D_{2k+1}|)$ queries. □

Refinement Pass

We can now present the full `Refine` method (Algorithm C.4), which simply scans over the layers and calls `RefineABA` on them. Our `Refine` is a modified version of `Refine` from [CLSSW19, Algorithm 11] using our new `RefineABA` method instead of just `RefineAB` and `RefineBA`. Just replacing the `Refine` method in the final algorithm

of [CLSSW19] with our modified `Refine` below leads to an $\tilde{O}(n\sqrt{r}/\varepsilon^{1.5})$-query algorithm (compared to their $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$), and concludes our first improvement (as discussed in Item 1 in Section C.1.1).

Algorithm C.4: `Refine`(k)

```
for k = ℓ, ℓ − 1, ℓ − 2, ..., 1, 0 do
    RefineABA(k)
```

The following Lemma C.3.12 will be useful to bound the number of `Refine` calls needed in our final algorithm, and closely corresponds to [CLSSW19, Corollary 43]. Our `Refine` implementation has the advantage that it only counts the elements in the **even** layers, of which there are at most r.

Lemma C.3.12. *Let $(B_1^{old}, A_1^{old}, \ldots)$ and $(B_1^{new}, A_1^{new}, \ldots)$ be the sets before and after `Refine` is run. Then at least $|B_1^{new}| - |B_{\ell+1}^{new}|$ elements in even layers have changed types.*

Proof. Note that whenever A_k changes, it is because some elements changed it types in D_{2k}. In particular, if the size of A_k increases (respectively decreases) by z, at least z elements will change types from fresh to selected (respectively from selected to removed) in D_{2k}.

After the first iteration $|A_\ell| = |B_{\ell+1}^{new}|$, so at least $|A_\ell^{old}| - |B_{\ell+1}^{new}|$ elements in $D_{2\ell}$ changed types. Similarly, after the iteration when $k = i$ (for $1 \le i \le \ell - 1$), $|A_i| = |A_{i+1}|$, and hence at least $|A_i^{old}| - |A_i|$ elements in D_{2i} changed types plus at least $|A_{i+1}| - |A_{i+1}^{old}|$ elements in D_{2i+2} changed types.[13] Finally, after the last iteration $|A_1| = |B_1^{new}|$, and hence at least $|B_1^{new}| - |A_1^{old}|$ elements in D_2 changed types.

The above terms telescope, and we conclude that at least $|B_1^{new}| - |B_{\ell+1}^{new}|$ elements in the even layers changed its types when `Refine` was run. □

Lemma C.3.13. *`Refine` uses $O(n)$ independence queries.*

Proof. This follows directly by Lemma C.3.11. □

C.3.2 Refining Along a Path

If we just run `Refine` until we get a maximal augment set (i.e. until $|B_1| = |B_{\ell+1}|$) we need to potentially run `Refine` as many as $\Theta(r)$ times, which needs too many independence queries. Lemma C.3.12 tells us that `Refine` makes the most "progress" while $|B_1| - |B_{\ell+1}|$ is large: in fact, only $O(r/p)$ calls to `Refine` is needed until $|B_1| - |B_{\ell+1}| \le p$. The idea in [CLSSW19] is thus to stop refining when $|B_1| - |B_{\ell+1}|$

[13] $|A_{i+1}| \le |A_{i+1}^{old}|$ just before the `RefineABA`(i) call, since earlier iterations can only have decreased the size of $|A_{i+1}|$.

is small enough and fall back to finding augmenting paths one at a time (they prove that one needs to find at most $O((|B_1| - |B_{\ell+1}|)\ell)$ many). We use a similar idea in that we swap to a different procedure when $|B_1| - |B_{\ell+1}|$ is small enough, the difference being that we still work with the partial augmenting set. This will let us show that only $O(|B_1| - |B_{\ell+1}|)$ many "paths" need to be found, saving a factor $\ell \approx \frac{1}{\varepsilon}$ compared to [CLSSW19].

This section thus describes the second improvement (as discussed in Item 2 in Section C.1.1). Note that this improvement is independent of the first improvement (i.e. the three-layer refine). We aim to prove the following lemma.

Lemma C.3.14. *There exists a procedure (`RefinePath`, Algorithm C.5), which uses $O(n \log r)$ independence queries, preserves the invariants, and either:*

i. Increases the size of $B_{\ell+1}$ by at least 1.

ii. Terminates with $(B_1, A_1, \ldots, B_{\ell+1})$ being a maximal augmenting set.

`RefinePath` attempts to find what we call a *valid path*. What we want is a sequence of elements which we can add to the partial augmenting set without violating the invariants and the properties of the partial augmenting set. It turns out (not very surprisingly) that such sequences of elements can be characterized by a notion of *paths* in something which resembles the *exchange graph with respect to our partial augmenting set*. This is what motivates the definition of *valid paths* below.

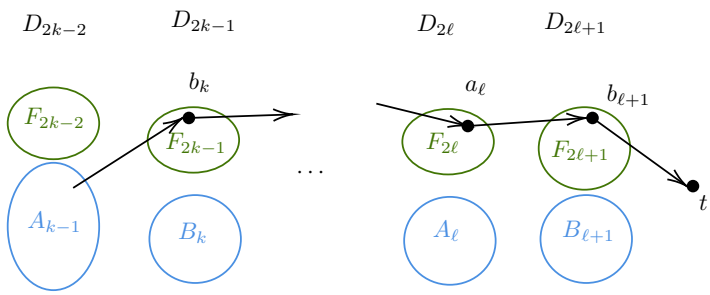


Figure C.3: A valid path $(b_k, \ldots, a_\ell, b_{\ell+1}, t)$ "starting" from the partial augmenting set at A_{k-1}, so that we can use Lemma C.3.17 and augment along it.

Definition C.3.15 (Valid path)**.** A sequence of elements $(b_i, a_i, b_{i+1}, \ldots, b_{\ell+1}, t)$ (or $(a_i, b_{i+1}, \ldots, b_{\ell+1}, t)$) is called a *valid path* (with respect to the partial augmenting set) if for all $k \geq i$:

(a) $a_k \in F_{2k}$ and $b_k \in F_{2k-1}$.

(b) $S + B_{\ell+1} + b_{\ell+1} \in \mathcal{I}_2$.

(c) $S - A_k + B_k - a_k + b_k \in \mathcal{I}_2$.

(d) $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$.

Remark C.3.16. Compare the properties of valid paths with the edges in the exchange graph from Definition C.2.5. A valid path is essentially a path in the exchange graph *after* we have already augmented S by our partial augmenting set (even though this exchange graph is not exactly defined, since it is not guaranteed that S remains a common independent set when augmented by a *partial* augmenting set).

Lemma C.3.17. *If $p = (b_i, a_i, b_{i+1}, \ldots, b_{\ell+1}, t)$ is a valid path starting at b_i, such that $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$, then $(B_1, A_1, \ldots, B_{i-1}, A_{i-1}, B_i + b_i, A_i + a_i, \ldots, B_{\ell+1} + b_\ell)$ is a partial augmenting set satisfying the invariants.*

Proof. That it forms a partial augmenting set is true by the definition of valid paths, and the fact that $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$. Indeed, it cannot be the case that $|A_{i-1}| < |B_i + b_i|$ when $i > 1$, since then $\text{rank}_1(S - A_{i-1} + B_i + b_i) > |S| = \text{rank}_1(S)$ implies that some element $x \in (B_i + b_i)$ satisfies $S + x \in \mathcal{I}_1$ (i.e. it is in the first layer D_1) by the exchange property of matroids. Invariants (c) and (d) are trivially true since the sets A_k and B_k are only extended. □

The goal of `RefinePath` (Algorithm C.5) is thus to find a valid path satisfying the conditions in Lemma C.3.17. Towards this goal, `RefinePath` will start from the last layer $D_{2\ell+1}$ and "scan left" in a breadth-first-search manner while keeping track of valid paths starting at each fresh vertex x (the next element on such a path will be stored as `next`$[x]$). If at some point one valid path can "enter" the partial augmenting set in a layer, we are done and can use Lemma C.3.17. We also show that it is safe (i.e. preserves the invariants) to remove all the fresh elements x for which we cannot find a valid path starting at x.

To efficiently find the "edges" during our breadth-first-search using only independence queries, we use the binary-search trick from Lemma C.2.8. However, this relies on the partial augmenting set being locally "flat" in the layers we are currently exploring, i.e. $|B_k| = |A_k|$ respectively $|B_k| = |A_{k+1}|$. We can ensure this by running `RefineAB` respectively `RefineBA` while performing the scan.

Now we are ready to present the pseudo-code of the `RefinePath` method (Algorithm C.5). Due to the asymmetry between even/odd layers and independence queries, we need to handle moving from layer B to A and from A to B a bit differently, but the ideas are similar.

Lemma C.3.18. *`RefinePath` preserves the invariants.*

Proof. The proof is relatively straightforward, but technical. The only non-trivial part is showing that invariants (c) and (d) are preserved after we remove something in line 8 or line 23. Intuitively, if we remove b in line 8, we can instead think of temporarily adding b to B_k and running `RefineBA`(k) in such a way so that b is

Algorithm C.5: `RefinePath`

```
1  for k = ℓ+1, ℓ, ..., 2, 1 do
       // Process (B_k, A_k)
2      RefineBA(k)
3      if some element a was added to A_k in the above refine-call then
4          Add the valid path starting at next[a] to the partial augmenting set
5          return
6      for each element b ∈ F_{2k-1} do
7          if S − A_k − F_{2k} + B_k + b ∉ I_2 then
8              Remove b, that is: F_{2k-1} ← F_{2k-1} − b,   R_{2k-1} ← R_{2k-1} + b
9          else
10             Find an a ∈ F_{2k} such that S − A_k + B_k + b − a ∈ I_2
11             Let next[b] = a
12             if k = ℓ+1 then
13                 Let next[b] = t
       // Process (A_{k-1}, B_k)
14     if some element b ∈ F_{2k-1} satisfies S − A_{k-1} + B_k + b ∈ I_1 then
15         Add the valid path starting at b to the partial augmenting set
16         return
17     RefineAB(k − 1)
18     Q ← F_{2k-2}
19     for each element b ∈ F_{2k-1} do
20         while can find a ∈ Q such that S − A_{k-1} + B_k + b − a ∈ I_1 do
21             Q ← Q − a
22             Let next[a] = b
23     Remove all elements in Q, that is:
         F_{2k-2} ← F_{2k-2} − Q,   R_{2k-2} ← R_{2k-2} + Q
   // Now (B_1, A_1, ..., B_{ℓ+1}) is a maximal augmenting set.
```

immediately removed. A similar intuitive argument works for line 23. We next present a formal proof.

We know that `RefineAB` and `RefineBA` preserve the invariants, by Lemma C.3.4. We also know by Lemma C.3.17 that adding a valid path to the partial augmenting set also preserves the invariants. So what remains is to show that the invariants are preserved after:

Line 8. We only need to check invariant (d), the other ones trivially hold. Let $W = S - A_k - F_{2k} + B_k = S - (D_{2k} - R_{2k}) + B_k$ and R^{old}_{2k-1} be R_{2k-1} before b was added to it. Note that b is such that $W + b \notin \mathcal{I}_2$, and we know

that $W \subseteq S - A_k + B_k \in \mathcal{I}_2$ and hence $\text{rank}_2(W + R^{old}_{2k-1}) = \text{rank}_2(W) = |W|$ and $\text{rank}_2(W + b) = \text{rank}_2(W) = |W|$. We thus need to show that $\text{rank}_2(W + R^{old}_{2k-1} + b) = |W|$ too, which is clear since W is a maximal independent subset of $W + R^{old}_{2k-1} + b$ (it can neither be extended with elements from R^{old}_{2k-1} nor with b).

Line 23. We only need to check invariant (c), the other ones trivially hold. We imagine we add the $a \in Q$ to R_{2k-2} one-by-one, and show that the invariant (c) is preserved after each such addition. So consider some $a \in Q$ which will be removed, and let R^{old}_{2k-2} be the set R_{2k-2} just before we added a to it. First note that $\text{rank}_1(S - A_{k-1} + B_k + F_{2k-1} - a) = \text{rank}_1(S - A_{k-1} + B_k + F_{2k-1}) - 1 = |S - A_{k-1} + B_k| - 1$, as otherwise there must exist some $b \in F_{2k-1}$ such that $S - A_{k-1} + B_k + b - a \in \mathcal{I}_1$ (by the matroid exchange property), and a would have been discovered in line 21 and therefore been removed from Q. So the "return" of adding a to $S - A_{k-1} + B_k + F_{2k-1} - a$ is increasing the rank by 1. Now consider some arbitrary $X \subseteq B_k + F_{2k-1}$ such that $S - A_{k-1} + X - R^{old}_{2k-2} - a \in \mathcal{I}_1$. We need to show that $S - A_{k-1} + X \in \mathcal{I}_1$. Note that $S - A_{k-1} + X - R^{old}_{2k-2} - a \subseteq S - A_{k-1} + B_k + F_{2k-1} - a$. Hence, by the diminishing returns (of adding a) we know $\text{rank}_1(S - A_{k-1} + X - R^{old}_{2k-2}) \geq \text{rank}_1(S - A_{k-1} + X - R^{old}_{2k-2} - a) + 1 = |S - A_{k-1} + X - R^{old}_{2k-2}|$, or equivalently that $S - A_{k-1} + X - R^{old}_{2k-2} \in \mathcal{I}_1$. Since the invariant held before, we conclude that $S - A_{k-1} + X \in \mathcal{I}_1$ too, which finishes the proof. □

Valid paths. The algorithm keeps track of a valid path starting at each fresh vertex it has processed. That is, after processing layer D_k, all elements in F_k must be the beginning of a valid path, else they were removed. In particular, the algorithm remembers the valid path starting at x as $(x, \texttt{next}[x], \texttt{next}[\texttt{next}[x]], \ldots)$. It is easy to verify that this sequence does indeed satisfy the conditions of *valid paths* by inspecting lines 10 and 21.

We also discuss what happens when the algorithm chooses to add a valid path to the partial augmenting set (i.e. in line 4 or 15). If we are in Line 15, we can directly apply Lemma C.3.17. Say we instead are in Line 4, and some a which was previously fresh has been added to A_k. The `RefineBA` call can only have increased A_k (that is $A_k \supseteq A^{old}_k + a$), so $S - A_k + B_{k+1} + b \in \mathcal{I}_1$ will holds for $b = \texttt{next}[a]$ and we can apply Lemma C.3.17 here too.

When no path is found. In the case when no valid path to add to the partial augmenting set is found, `RefinePath` must terminate with $|B_1| = |A_1| = \cdots = |B_{\ell+1}|$. This is because the `RefineAB` and `RefineBA` will never select any new elements. That is `RefineBA` will not change A_k (as otherwise we enter the if-statement at line 4), and `RefineAB` will not change B_k (since if $b \in F_{2k-1}$ with $S - A_{k-1} + B_k + b \in \mathcal{I}_1$ existed we would have entered the if-statement at line 15). We also remark that `RefinePath` ends with B_1 being a maximal subset of $D_1 \setminus R_1$,

as otherwise some b would have been found in line 14. Hence Lemma C.3.3 implies that $(B_1, A_1, \ldots, B_{\ell+1})$ now forms a *maximal* augmenting set.

Query complexity. The `RefineAB` and `RefineBA` calls will in total use $O(n)$ queries. The independence checks at Lines 7 and 14 happens at most once for each element, and thus use $O(n)$ queries in total. Lines 10 and 21 can be implemented using the binary-search-exchange-discovery Lemma C.2.8. Hence Line 10 will use, in total, $O(n \log r)$ queries and Line 21 will use, in total, $O(n \log r)$ queries (since each $a \in Q$ will be discovered at most once). So we conclude that Algorithm C.5 uses $O(n \log r)$ independence queries.

C.3.3 Hybrid Algorithm

Now we are finally ready to present the full algorithm of a phase, which is parameterized by a variable p. The following algorithm is similar to that of [CLSSW19, Algorithm 12] but uses our improved `Refine` method and finds individual paths using the `RefinePath` method.

Algorithm C.6: Phase ℓ

1 Calculate the distance layers by a BFS.
2 Run `Refine` (Algorithm C.4) until $|B_1| - |B_{\ell+1}| \leq p$, but at least once.
3 Run `RefinePath` (Algorithm C.5) until $(B_1, A_1, \ldots B_{\ell+1})$ is maximal and augment along it.

Lemma C.3.19. *Except for line 1, Algorithm C.6 uses $O(nr/p + np \log r)$ queries.*[14]

Proof. Lemma C.3.12 tells us that `Refine` changes types of at least p elements in even layers (i.e. elements in S) every time it is run, except maybe the last time. Thus we only run `Refine` $O(|S|/p + 1)$ times. Each call takes $O(n)$ queries (Lemma C.3.13), for a total of $O(nr/p)$ queries in line 2 of the algorithm.

Now we argue that B_1 can never become larger than what it was just after line 2 was run. This is because `Refine` will run at least once, and ends with a `RefineABA`(0) call which in turn ends with a `RefineAB(0)` call—which extends B_1 to be a maximal set in $D_1 \setminus R_1$ for which $S + B_1 \subseteq \mathcal{I}_1$ holds.[15]

Lemma C.3.14 tells us that each (except the last) time `RefinePath` is run, $B_{\ell+1}$ increases by 1. This can happen at most p times, so line 3 uses a total of $O(np \log r)$ queries. □

[14] Compare this to $O(n^2/p + np\ell \log r)$ in [CLSSW19]. The improvement from n^2/p to nr/p comes from the use of the new three-layer `RefineABA` method, and the (independent) improvement from $np\ell \log r$ to $np \log r$ comes from the use of the new `RefinePath` method.

[15] Indeed, since $\mathcal{M}_1$ is a matroid, all such maximal sets have the same size, so we can never obtain something larger later.

Now it is easy to prove Theorem C.1.1, which we restate below.

Theorem C.1.1 (Approximation algorithm)**.** *There is a deterministic algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (1-\varepsilon)r$, using $O\left(\frac{n\sqrt{r \log r}}{\varepsilon}\right)$ independence queries.*

Proof. Pick $p = \sqrt{r/\log r}$.[16] Then each phase will use $O(n\sqrt{r \log r})$ independence queries (by Lemma C.3.19), plus a total of $O(\frac{1}{\varepsilon} n \log r)$ to run the BFS's across all phases (see [CLSSW19] for details on the BFS implementation). Since we need only run $O(\frac{1}{\varepsilon})$ phases (by Lemma C.2.7 and Theorem C.2.13), in total the algorithm will use $O(\frac{1}{\varepsilon} n\sqrt{r \log r})$ queries. □

C.4 Exact Matroid Intersection

In this section, we prove Theorem C.1.2 (restated below) by showing how our improved approximation algorithm leads to an improved exact algorithm when combined with the algorithms of [BBMN21].

Theorem C.1.2 (Exact algorithm)**.** *There is a randomized algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum cardinality r, and w.h.p.[17] uses $O(nr^{3/4} \log n)$ independence queries. There is also a deterministic exact algorithm using $O(nr^{5/6} \log n)$ queries.*

Approximation algorithms are great at finding the *many, very short* augmenting paths efficiently. Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai [BBMN21, Algorithm 2] very recently showed how to efficiently find the remaining *few, very long* augmenting paths, with a randomized algorithm using $\tilde{O}(n\sqrt{r})$ queries per augmentation (or, with a slightly less efficient deterministic algorithm using $\tilde{O}(nr^{2/3})$ queries). In the randomized $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of [BBMN21, Algorithm 3], the current bottleneck is the approximation algorithm used. Replacing the use of the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query approximation algorithm from [CLSSW19] with our improved version we obtain the more efficient randomized[18] $\tilde{O}(nr^{3/4})$-query Algorithm C.7.

[16] Compare this to $p = \sqrt{n\varepsilon/\log r}$ in [CLSSW19].

[17] *w.h.p. = with high probability* meaning with probability $1 - n^{-c}$ for some arbitrarily large constant c.

[18] The deterministic algorithm of Theorem C.1.2 is obtained in the same fashion but by using the deterministic version of the augmenting path finding algorithm [BBMN21, Algorithm 2].

Algorithm C.7: Exact Matroid Intersection (Modified version of [BBMN21, Algorithm 3])

1 Run the approximation algorithm (Theorem C.1.1) with $\varepsilon = r^{-1/4}$ to obtain a common independent set S of size at least $(1-\varepsilon)r = r - r^{3/4}$.
2 Starting with S, run Cunningham's algorithm (as implemented by [CLSSW19]), until the distance between s and t becomes larger than $r^{3/4}$.
3 Keep finding augmenting paths—one at a time—to augment along, using the randomized $O(n\sqrt{r}\log n)$-query algorithm of [BBMN21, Algorithm 2]. When no (s,t)-path can be found in the exchange graph, S is a largest common independent set.

Query complexity. We analyse the individual lines of Algorithm C.7.

Line 1. We see that the approximation algorithm uses $O(nr^{3/4}\log n)$ queries in line 1.

Line 2. One need to (i) compute distances up to $d = r^{3/4}$, and (ii) perform at most $O(r^{3/4})$ augmentations. [CLSSW19; BBMN21; Ngu19] show how to do (i) in $O(nd\log n) = O(nr^{3/4}\log n)$ queries in total over all phases of Cunningham's algorithm, and how to do (ii) using $O(n\log n)$ queries per augmentation (for a total of $O(nr^{3/4}\log n)$ queries).

Line 3. By Lemma C.2.7, line 3 runs $O(r^{1/4})$ times—each using $O(n\sqrt{r}\log n)$ queries—for a total of $O(nr^{3/4}\log n)$ queries.

Remark C.4.1. In Algorithm C.7, the bottleneck between line 1-2 and line 2-3 now matches (which was not the case in [BBMN21]). This means that if one wants to improve the algorithm by replacing the subroutines in line 1 and 3, one need to **both** improve the approximation algorithm (line 1) and the method to find a single augmenting-path (line 3).

Acknowledgement

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 71567.

I also want to thank Danupon Nanongkai and Sagnik Mukhopadhyay for insightful discussions and their valuable comments throughout the development of this work.

Bibliography

[AD71] Martin Aigner and Thomas A. Dowling. "Matching Theory for Combinatorial Geometries". In: *Transactions of the American Mathematical Society* 158.1 (1971), pp. 231–245 (cit. on pp. 221, 224).

[BBMN21] Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Breaking the Quadratic Barrier for Matroid Intersection". In: *STOC*. ACM, 2021 (cit. on pp. 221–224, 240, 241).

[CLSSW19] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. "Faster Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2019, pp. 1146–1168 (cit. on pp. 221–230, 233–235, 239–241).

[CQ16] Chandra Chekuri and Kent Quanrud. "A Fast Approximation for Maximum Weight Matroid Intersection". In: *SODA*. SIAM, 2016, pp. 445–457 (cit. on p. 221).

[Cun86] William H. Cunningham. "Improved Bounds for Matroid Partition and Intersection Algorithms". In: *SIAM J. Comput.* 15.4 (1986), pp. 948–957 (cit. on pp. 221, 222, 224, 225).

[Edm70] Jack Edmonds. "Submodular functions, matroids, and certain polyhedra". In: *Combinatorial structures and their applications*. 1970, pp. 69–87 (cit. on p. 221).

[Edm79] Jack Edmonds. "Matroid intersection". In: *Annals of discrete Mathematics*. Vol. 4. Elsevier, 1979, pp. 39–49 (cit. on pp. 221, 224).

[EDVJ68] Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. "Matroid partition". In: *50 Years of Integer Programming 1958–2008* (1968), p. 199 (cit. on p. 221).

[Law75] Eugene L. Lawler. "Matroid intersection algorithms". In: *Math. Program.* 9.1 (1975), pp. 31–56 (cit. on pp. 221, 224).

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. "A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization". In: *FOCS*. IEEE Computer Society, 2015, pp. 1049–1065 (cit. on p. 221).

[Ngu19] Huy L. Nguyen. "A note on Cunningham's algorithm for matroid intersection". In: *CoRR* abs/1904.04129 (2019) (cit. on pp. 221, 224, 225, 241).

Paper D

Sublinear-Round Parallel Matroid Intersection

Joakim Blikstad

Article published in 49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France, pp. 25:1–25:17. [Bli22]
Full version at `https://doi.org/10.4230/LIPIcs.ICALP.2022.25`.

Abstract

Despite a lot of recent progress in obtaining faster sequential matroid intersection algorithms, the fastest *parallel* poly(n)-query algorithm was still the straightforward $O(n)$-round parallel implementation of Edmonds' augmenting paths algorithm from the 1960s.

Very recently, Chakrabarty-Chen-Khanna [FOCS'21] showed the lower bound that any, possibly randomized, parallel matroid intersection algorithm making poly(n) rank-queries requires $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity. They ask, as an open question, if the lower bound can be improved to $\tilde{\Omega}(n)$, or if there can be sublinear-round, poly(n)-query algorithms for matroid intersection.

We resolve this open problem by presenting the first sublinear-round parallel matroid intersection algorithms. Perhaps surprisingly, we do not only break the $\tilde{O}(n)$-barrier in the rank-oracle model, but also in the weaker independence-oracle model. Our rank-query algorithm guarantees $O(n^{3/4})$ rounds of adaptivity, while the independence-query algorithm uses $O(n^{7/8})$ rounds of adaptivity, both making a total of poly(n) queries.

D.1 Introduction

Matroid intersection. Given two matroids[1] $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same n-element ground set V (but with different notions of independence $\mathcal{I}_1, \mathcal{I}_2$), the *matroid intersection problem* is to find the largest common independent set $S^* \subseteq \mathcal{I}_1 \cap \mathcal{I}_2$. This is a fundamental discrete optimization problem that has been studied for over half a century. Matroid intersection can be used to model many important combinatorial optimization problems, such as bipartite matching, finding arborescences, spanning tree packing, etc. As such, matroid intersection is a natural avenue to study all these problems simultaneously.

Oracle access. There are two standard ways to access the matroids—*independence oracles* and *rank oracles*—and we study both in this work. In an *independence-query* we may ask if $S \subseteq V$ is independent in one of the matroids, i.e. a query of the form "Is $S \in \mathcal{I}_1$?" or "Is $S \in \mathcal{I}_2$?" In a *rank-query* we instead ask for the *rank* of $S \subseteq V$ in one of the matroids. The rank $\text{rank}_1(S)$ with respect to the matroid $\mathcal{M}_1$ (similarly rank_2 for $\mathcal{M}_2$) is the size of the largest (or, equivalently, any maximal) independent set, w.r.t. $\mathcal{I}_1$, contained in S. Note that the rank oracle is strictly more powerful than the independence oracle, since $S \in \mathcal{I}_1$ if and only if $\text{rank}_1(S) = |S|$.

Parallel matroid intersection. A parallel matroid intersection algorithm accesses the oracle in rounds. In each round, a number of queries—that may only depend on the answers to queries made in previous rounds—can be issued in parallel. There is certainly a trade-off between (1) *adaptivity*, usually measured by the number of rounds, and (2) the total number of queries. When constructing parallel algorithms the goal is often to have as few rounds of adaptivity as possible while making only polynomially many queries in total.

Previous work. Edmonds [EDVJ68] showed the first polynomial algorithm for matroid intersection in the 1960s, using $O(n^3)$ independence-queries, and there has been a long line of research since then e.g. [EDVJ68; Edm70; Edm79; AD71; Law75; Cun86; LSW15; Ngu19; CLSSW19; BBMN21; Bli21]. Many of these are based on Edmonds' framework of finding *augmenting paths* in the *exchange graph*. In the sequential setting, only recently was the quadratic $O(n^2)$-query-barrier broken, first for rank-queries by Chakrabarty-Lee-Sidford-Singla-Wong in FOCS 2019 [CLSSW19] and subsequently also for independence-queries by Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai in STOC 2021 [BBMN21]. The current state-of-the-art in the sequential setting are the[2] $\tilde{O}(n\sqrt{n})$ rank-query algorithm by [CLSSW19] and the $\tilde{O}(n^{7/4})$ independence-query algorithm by [Bli21].

[1] Matroids are a well-studied combinatorial structure which can be though of as a generalization of the notion of linear independence in vector spaces. For a formally definition, see Definition D.2.2.

[2] We use the usual convention of hiding polylog(n)-factors with $\tilde{O}$ and $\tilde{\Omega}$ throughout the paper.

When it comes to the *parallel* setting, there is a straightforward $O(n)$-round, $\text{poly}(n)$ independence-query implementation of Edmonds' algorithm: find the (up to $O(n)$ many) augmenting paths one-by-one. Each augmenting path can be found in a single round by querying all the potential edges in the exchange graph. In some special cases of matroid intersection we can do much better: a sequence of work has shown that both bipartite matching [Lov79; KUW86; FGT21] and subsequently linear matroid intersection [Lov79; GT20] are in RNC[3] and quasi-NC.[4]

Another line of relevant work is showing that, in the parallel setting, the *search*-problem (finding a largest common independent set S) and the *decision*-problem (just finding the size of the answer) are "equivalent" (with only $O(\text{polylog}(n))$ overhead). This is not at all obvious in the parallel setting, however, a recent work from SODA 2022 by Ghosh-Gurjar-Ray [GGR22] shows that this is indeed the case for *weighted* matroid intersection, with rank-oracle access.

In FOCS 1985, Karp-Upfal-Wigderson [KUW85] showed that any *independence-query* algorithm, possibly randomized, that finds a maximum independent set (basis) in a *single* matroid must use $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity if it makes $\text{poly}(n)$ queries. They also show algorithms to find a basis of a (single) matroid in $O(\sqrt{n})$ rounds of independence-queries or a single round of the more powerful rank-queries. Arguably, this polynomial gap between the independence-query ($\tilde{\Omega}(n^{1/3})$ rounds) and rank-query ($O(1)$ rounds) for the seemingly easy problem to find a basis of a matroid illustrates that the independence-query is much weaker than the rank-query when used in parallel algorithms.

Nevertheless, a recent result from FOCS 2021 by Chakrabarty-Chen-Khanna [CCK21] shows that even *rank-query* algorithms require a polynomial number of rounds to solve matroid intersection. In particular, they show a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity for any, possibly randomized, $\text{poly}(n)$ rank-query matroid intersection algorithm.

Despite efficient algorithms for some special cases of matroid intersection, the trivial $O(n)$-round algorithm has remained unbeaten in the general case. The major open question (asked, for example, by [CCK21]) is then whether it is possible to beat the $O(n)$-round barrier, or if matroid intersection is inherently very sequential and requires $\tilde{\Omega}(n)$ rounds of adaptivity.

Our results. We answer the above question by showing the first sublinear-round parallel matroid intersection algorithms, both in the rank-oracle and independence-oracle models. In particular, we obtain the following theorem.

Theorem D.1.1 (Sublinear-round Matroid Intersection)**.** *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(n^{3/4})$ rounds of polynomially many rank-queries, or*

[3] Randomized $\text{polylog}(n)$ rounds of adaptivity with $\text{poly}(n)$ total work.
[4] Deterministic $\text{polylog}(n)$ rounds of adaptivity with $n^{O(\log n)}$ total work.

- $O(n^{7/8})$ *rounds of polynomially many independence-queries.*

Our results, together with the lower bounds of [KUW85; CCK21], imply that the true adaptivity of matroid intersection is somewhere between $n^{1/3}$ and $n^{3/4}$ (or $n^{7/8}$ for independence queries).

Remark D.1.2. Although we focus on the query-complexity in this paper, we note that the rounds and work in our algorithms are dominated, up to log-factors, by the oracle queries.

D.1.1 Technical Overview

The exchange graph and augmenting paths. Like many matroid intersection algorithms, we work in Edmonds' framework of finding *augmenting paths* in the *exchange graph*. The *exchange graph* $G(S)$ with respect to a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a directed bipartite graph, where finding a shortest (s,t)-path—called an *augmenting path*—means that we can increase the size of S by one. In a single round of $O(n^2)$ independence (or rank) queries, we can learn the entire exchange graph, and can thus find an augmenting path if one exists. This immediately gives a straightforward $O(n)$-round algorithm: *find the (up to $O(n)$ many) augmenting paths one-by-one.*

The exchange graph depends on the current common independent set S, and changes after each augmentation. In fact, if we have two disjoint augmenting paths p_1 and p_2 in $G(S)$, it is **not** necessarily the case that we can augment along both of these: augmenting along p_1 might destroy the path p_2 even if they were disjoint.[5] This forms the main difficulty in trying to beat the $O(n)$-round barrier, and illustrates the need in finding several "compatible" augmenting paths which can all be augmented along simultaneously.

Blocking flow. Cunningham [Cun86] was the first to introduce *blocking flow* algorithms to matroid intersection, similar to Hopcroft-Karp's [HK73] bipartite matching or Dinitz's [Din70] max-flow algorithms. The idea is to run in phases, where after each phase the length of a shortest augmenting path in the exchange graph has increased. This is done by finding a maximal collection of compatible shortest augmenting paths. Both of the current state-of-the-art sequential $O(n\sqrt{n})$-rank-query [CLSSW19] and $O(n^{7/4})$-independence-query [Bli21] algorithms are based on versions of these blocking flow ideas. The $O(n\sqrt{n})$-rank-query algorithm still finds the augmenting paths in a sequential way, so it does unfortunately not seem to parallelize well.

The $O(n^{7/4})$-independence-query algorithm, on the other hand, is based on a recent notion of *augmenting sets* introduced by Chakrabarty-Lee-Sidford-Singla-Wong [CLSSW19]. This notion of *augmenting sets* precisely captures what a collection of

[5] This is unlike the case of augmenting path algorithms for bipartite matching or maximum flow, where one can indeed augment along disjoint paths simultaneously.

"compatible" shortest augmenting paths looks like. The authors of [CLSSW19] also present an algorithm to find such augmenting sets, using independence-queries.

Our contribution is to show that a modified version of the augmenting sets algorithm of [CLSSW19, Section 6] (which was later improved by [Bli21]) can be implemented in parallel when combined with the parallel matroid-basis finding algorithms of Karp-Upfal-Wigderson [KUW85]. Previous to this work, augmenting sets algorithms have before only been used in the sequential setting, and only in the independence-oracle model. Nevertheless, augmenting sets are what allows us to break the $O(n)$-round barrier also with rank-queries.

D.2 Preliminaries

We use the standard definitions of *matroid* $\mathcal{M} = (V, \mathcal{I})$; *rank* $\text{rank}(X)$ for any $X \subseteq V$; *exchange graph* $G(S)$ for a *common independent set* $S \in \mathcal{I}_1 \cap \mathcal{I}_2$; and *augmenting paths* in $G(S)$ throughout this paper. For completeness, we define them below. We also need the notions of *augmenting sets* introduced by [CLSSW19], which we also define in later this section.

Definition D.2.1 (Set notation)**.** We will use $A + x$ and $A - x$ to denote $A \cup \{x\}$ respectively $A \setminus \{x\}$, as is usual in matroid intersection literature. We will also use $A + B := A \cup B$, and $A - B := A \setminus B$.

Matroids

Definition D.2.2 (Matroid)**.** A *matroid* is a tuple $\mathcal{M} = (V, \mathcal{I})$ of a *ground set* V of n elements, and non-empty family $\mathcal{I} \subseteq 2^V$ of *independent sets* satisfying:

Downward closure: if $S \in \mathcal{I}$, then $S' \in \mathcal{I}$ for all $S' \subseteq S$.

Exchange property: if $S, S' \in \mathcal{I}$, $|S| > |S'|$, then there exists $x \in S \setminus S'$ such that $S' + x \in \mathcal{I}$.

Definition D.2.3 (Matroid rank)**.** The *rank* of $A \subseteq V$, denoted by $\text{rank}(A)$, is the size of the largest (or, equivalently, any maximal) independent set contained in A. It is well-known that the rank-function is submodular, i.e. $\text{rank}(A+x) - \text{rank}(A) \geq \text{rank}(B+x) - \text{rank}(B)$ whenever $A \subseteq B \subseteq V$ and $x \in V \setminus B$. Note that $\text{rank}(A) = |A|$ if and only if $A \subseteq \mathcal{I}$.

Definition D.2.4 (Matroid Intersection)**.** Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set V, a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. The *matroid intersection problem* asks us to find the largest common independent set. We use rank_1 and rank_2 to denote the rank functions of the corresponding matroids, and $n = |V|$ to be the size of the ground set.

The Exchange Graph

Definition D.2.5 (Exchange graph)**.** Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set, and a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph on vertex set $V \cup \{s, t\}$ with the following arcs (or directed edges):

1. (s, b) for $b \in V \setminus S$ when $S + b \in \mathcal{I}_1$.
2. (b, t) for $b \in V \setminus S$ when $S + b \in \mathcal{I}_2$.
3. (a, b) for $b \in V \setminus S, a \in S$ when $S - a + b \in \mathcal{I}_1$.
4. (b, a) for $b \in V \setminus S, a \in S$ when $S - a + b \in \mathcal{I}_2$.

We will denote the set of elements at distance k from s by the distance-layer D_k. Note that $D_k \subseteq V \setminus S$ when k is odd and $D_k \subseteq S$ when k is even.

Definition D.2.6 (Shortest augmenting path)**.** In $G(S)$, a shortest (s, t)-path $p = (s, b_1, a_2, b_3, a_4, \ldots, a_{\ell-1}, b_\ell, t)$ (with $b_i \in V \setminus S$ and $a_i \in S$) is called a *shortest augmenting path*. We can *augment* S along the path p to obtain $S' = S + b_1 - a_2 + b_3 - a_4 \ldots + b_\ell$, which is well-known to also be a common independent set (with $|S'| = |S| + 1$). Conversely, there must exist a shortest augmenting path whenever S is not a largest common independent set.

Augmenting Sets Augmenting Sets is a notion capturing a "blocking flow" in the exchange graph, and was introduced by [CLSSW19], and also subsequently used in the algorithms of [BBMN21; Bli21]. In order to efficiently find "good" augmenting sets, the algorithm works with a relaxed form of them instead, called *partial* augmenting sets. The following definitions and key properties of (partial) augmenting sets are copied from [CLSSW19] where one can find the corresponding proofs.

Definition D.2.7 (Augmenting Sets, from [CLSSW19, Definition 24])**.** Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $G(S)$ be the corresponding exchange graph with shortest (s, t)-path of length $\ell + 1$ (ℓ must be odd) and distance layers $D_1, D_2, \ldots, D_\ell$. A collection of sets $\Pi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$[6] form an *augmenting set* in $G(S)$ if the following conditions are satisfied:

(a) $A_k \subseteq D_k$ for even k, and $B_k \subseteq D_k$ for odd k.

(b) $|B_1| = |A_2| = |B_3| = \cdots = |B_\ell|$

(c) $S + B_1 \in \mathcal{I}_1$

(d) $S + B_\ell \in \mathcal{I}_2$

[6] Our indexing of the sets differ a bit from [CLSSW19; Bli21].

(e) For all even $1 \leq k \leq \ell$, we have $S - A_k + B_{k+1} \in \mathcal{I}_1$

(f) For all odd $1 \leq k \leq \ell$, we have $S - A_{k+1} + B_k \in \mathcal{I}_2$

Definition D.2.8 (Partial Augmenting Sets, from [CLSSW19, Definition 37])**.** We say that $\Phi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$ forms a *partial augmenting set* if it satisfies the conditions (a), (c),[7] and (e) of an *augmenting set*, plus the following two relaxed conditions :

(b) $|B_1| \geq |A_2| \geq |B_3| \geq \cdots \geq |B_\ell|$.

(f) For all odd $1 \leq k \leq \ell$, we have $\text{rank}_2(S - A_{k+1} + B_k) = \text{rank}_2(S)$.

Theorem D.2.9 (from [CLSSW19, Theorem 25])**.** *Let* $\Pi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$ *be the an augmenting set in the exchange graph* $G(S)$*. Then the set* $S' := S \oplus \Pi_\ell := S + B_1 - A_2 + B_3 - \cdots - A_{\ell-1} + B_\ell$ *is a common independent set.*[8]

We also need the notion of *maximal* augmenting sets, which naturally correspond to a maximal ordered collection of shortest augmenting paths, where, after augmentation, the (s,t)-distance must have increased. Together with a lemma from [Cun86] (Lemma D.2.12), we can see, on a high-level, how to obtain $(1-\varepsilon)$-approximation algorithms: *find "blocking flows" (i.e. maximal augmenting sets) until the* (s,t)*-distance is* $\Omega(1/\varepsilon)$.

Definition D.2.10 (Maximal Augmenting Sets, from [CLSSW19, Definition 35])**.** Let $\Pi_\ell = (B_1, A_2, B_3, \cdots, A_{\ell-1}, B_\ell)$ and $\tilde{\Pi}_\ell = (\tilde{B}_1, \tilde{A}_2, \tilde{B}_3, \cdots, \tilde{A}_{\ell-1}, \tilde{B}_\ell)$ be two augmenting sets in $G(S)$. We say $\tilde{\Pi}_\ell$ *contains* Π_ℓ if $B_k \subseteq \tilde{B}_k$ and $A_k \subseteq \tilde{A}_k$, for all k. An augmenting set Π_ℓ is called *maximal* if there exists no other augmenting set $\tilde{\Pi}_\ell$ containing Π_ℓ.

Lemma D.2.11 (from [CLSSW19, Theorem 36])**.** *An augmenting set* Π_ℓ *is maximal if and only if there is no augmenting path of length at most* $\ell + 1$ *in* $G(S \oplus \Pi_\ell)$.

Lemma D.2.12 (Cunningham [Cun86])**.** *If the length of the shortest* (s,t)*-path in* $G(S)$ *is at least* $2\ell + 1$*, then* $|S| \geq (1 - O(1/\ell))r$*, where* r *is the size of the largest common independent set.*

D.3 Warm-up: Finding a *Maximal* Common Independent Set

Consider first the easier problem of finding a *maximal* (instead of *maximum*) common independent set: that is we want to find a set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ such that there is no

[7] Note that we intentionally skip item (d), unlike [CLSSW19] which includes it in the definition, however they do not always maintain this property in their algorithms.

[8] Note that $|S'| = |S| + |B_1|$. In particular, an augmenting set with $|B_1| = 1$ is exactly an augmenting path. [CLSSW19] shows that augmenting sets correspond exactly to a sequence of consecutive shortest augmenting paths.

$x \in V \setminus S$ for which $S + x \in \mathcal{I}_1 \cap \mathcal{I}_2$. It is well-known that a maximal common independent set is also a $\frac{1}{2}$-approximation for the matroid intersection problem, and indeed our algorithm for a general $(1-\varepsilon)$-approximation (Section D.4) will use similar ideas as our algorithm to find a *maximal* common independent set in this section.

In the sequential setting there is a very easy $O(n)$-query greedy algorithm: *Start with $S = \emptyset$ and go through all elements $x \in V$ and add them to S if $S + x$ is independent in both matroids.* However, this greedy algorithm is inherently very sequential and does not seem to adapt well to the parallel setting. Instead, we must somehow try to find several xs "in parallel" which we can all add to S simultaneously without breaking independence.

D.3.1 One Matroid

Let us start even simpler, and consider how to find a *maximal* independent set[9] S in a single matroid $\mathcal{M} = (V, \mathcal{I})$. It turns out that in our final matroid intersection algorithm we will many times, as a subroutinue, need to do exactly this.

Karp-Upfal-Wigderson [KUW85] provides some simple parallel algorithms (both for rank- and independence-oracle access), whose results we present in Lemma D.3.1. We briefly sketch their algorithms below, more details and full proofs can be found in [KUW85; KUW88].

Rank Oracle. The rank-query algorithm only needs a single round. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the elements of the ground set, and let $V_i = \{v_1, v_2, \ldots, v_{i-1}, v_i\}$ (so that $V_0 = \emptyset$ and $V_n = V$). Now query $\text{rank}(V_i)$ for all i, and return $S = \{v_i : \text{rank}(V_i) > \text{rank}(V_{i-1})\}$. Intuitively, we can imagine that we go through all elements v_i one-by-one and add them to S if and only if the rank goes up.

Independence Oracle. The independence-query algorithm will need $O(\sqrt{n})$ rounds of $O(n)$ queries per round. Partition the elements of V into $\sqrt{n}$ different groups of (almost) equal size $F_1, F_2, \ldots F_{\sqrt{n}}$. If any group is independent (say F_i), then we select it, and consider the contracted matroid $\mathcal{M}/F_i$. Note that this can only happen $\sqrt{n}$ times. On the other hand, if all F_i are dependent, then we will find one element per group (that is $\sqrt{n}$ in total) which we can safely discard: If $\{v_1, v_2, \ldots, v_k\} = F_i$ are the elements of F_i, we query all prefixes, i.e. "Is $\{v_1, v_2, \ldots, v_j\} \in \mathcal{I}$?" for all j, and discard the first element v_j for which the answer is "No".

Lemma D.3.1 (Parallel basis algorithm, [KUW85])**.** *There is a deterministic parallel algorithm which given a matroids $\mathcal{M} = (V, \mathcal{I})$ finds a **maximal** independent set $S \in \mathcal{I}$ using either*

[9] Such a set S is usually called a *basis* of the matroid, and due to the exchange-property all the maximal independent sets must have the same size.

- $O(1)$ *round of* $O(n)$ *many rank-queries, or*
- $O(\sqrt{n})$ *rounds of* $O(n)$ *many independence-queries.*[10]

Remark D.3.2. Note that if we have a set $X \in \mathcal{I}$ and $Y \subseteq V \setminus X$, we can with the same algorithms as above find a maximal $Y' \subseteq Y$ such that $X + Y' \in \mathcal{I}$, even though the above algorithms are only stated as if $X = \emptyset$ and $Y = V$. This is since we can consider the contracted and restricted matroid $\mathcal{M}' = (\mathcal{M} \,/\, X) \setminus (V \setminus Y)$; and an independence/rank-query on $\mathcal{M}'$ can be simulated with the corresponding query on $\mathcal{M}$.

D.3.2 Two Matroids

Now we return to our problem of finding a maximal common independent set S of two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$. Suppose we already have some common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. We will try to add more elements to S until it becomes maximal.

Firstly, let us concentrate on the first matroid and pick a maximal set $B \subseteq V$ such that $S + B \in \mathcal{I}_1$ using Lemma D.3.1. However, $S + B$ is not necessarily independent in the second matroid, so we would need to fix this: let $B' \subseteq B$ be a maximal subset such that $S + B' \in \mathcal{I}_2$, which we again can find using Lemma D.3.1. Now we know $S + B' \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a common independent set, so we set $S \leftarrow S + B'$, and we have made some progress (unless $B' = \emptyset$ of course).

At this point we can make a crucial observation: *we can safely discard the elements* $x \in B \setminus B'$*, since now* $S + x \notin \mathcal{I}_2$. Hence, for each element in B we have either (i) added it to our common independent set or (ii) discarded it. As long as $|B|$ is relatively large (say $\approx \sqrt{n}$), we have made significant progress.

On the other hand, if $|B|$ is small, we may resort to a different strategy. By the exchange property of matroids, we know that any $A \subseteq V$ such that $S + A \in \mathcal{I}_1$ has size $|A| \leq |B|$. So we can add at most $|B|$ more elements to our common independent set S before it becomes maximal. We can thus simply find these remaining (up to $|B|$ many) elements one-by-one, using one round each.

We present this two-stage strategy below in Algorithm D.1, which is parametrized by the cut-off threshold p for when to consider $|B|$ small. The optimal choice of p differs depending on the oracle access (independence or rank) we have.

Adaptivity. The first stage of Algorithm D.1 runs in $O(n/p \cdot \mathcal{T}_{\text{basis}})$ rounds if $\mathcal{T}_{\text{basis}}$ is the number of rounds needed to find a maximal independent set for a single matroid (Lemma D.3.1 gives $\mathcal{T}^{\text{rank}}_{\text{basis}} = O(1)$ and $\mathcal{T}^{\text{indep}}_{\text{basis}} = O(\sqrt{n})$). This is since the size of F will decrease by $|B| \geq p$ each time the while-loop is run, which can happen at most n/p times. The second stage of Algorithm D.1 runs in $O(p)$ rounds,

[10] KUW [KUW85] also provides a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds for any independence-query algorithm which uses only polynomial number of queries per round, even if randomization is allowed. It remains an open problem to close this gap between $\tilde{\Omega}(n^{1/3})$ and $O(\sqrt{n})$.

Algorithm D.1: Maximal Common Independent Set

Input: V: Set of elements, $\mathcal{I}_1$: Independent set structure 1, $\mathcal{I}_2$: Independent set structure 2
Output: S: Maximal common independent set
Data: $S = \emptyset$, $F = V$

while *true* **do**
  // Stage 1
  Find a maximal $B \subseteq F$ such that $S + B \in \mathcal{I}_1$
  if $|B| \leq p$ **then**
    break
  Find a maximal $B' \subseteq B$ such that $S + B' \in \mathcal{I}_2$
  Update $S \leftarrow S + B'$
  Update $F \leftarrow F - B$
while *true* **do**
  // Stage 2
  Query "Is $S + x \in \mathcal{I}_1$" and "Is $S + x \in \mathcal{I}_2$?" for all $x \in F$ in parallel
  Pick an arbitrary $x \in F$ such that $S + x \in \mathcal{I}_1$ and $S + x \in \mathcal{I}_2$
  if *no such x exists* **then**
    break
  Update $S \leftarrow S + x$
  Update $F \leftarrow F - x$

both for independence and rank-oracle. Picking p optimally gives: $O(\sqrt{n})$ rounds of rank-queries (with $p = \sqrt{n}$); or $O(n^{3/4})$ rounds of independence-queries (with $p = n^{3/4}$). This proves Theorem D.3.3, stated below.

Theorem D.3.3. *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a **maximal** common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(\sqrt{n})$ rounds of polynomially many rank-queries, or*
- *$O(n^{3/4})$ rounds of polynomially many independence-queries.*

D.4 Finding a *Maximum* Common Independent Set

In this section we present our sublinear-round matroid intersection algorithm.

The algorithm consists of two steps: first it finds an $(1 - \varepsilon)$-approximation, and then it finds the remaining εn (which is sublinear if $1/\varepsilon$ is polynomially large in n) augmenting paths one-by-one. Each such remaining augmenting path can be found in a single round of n^2 independence (or rank) queries: in parallel query each possible edge of the exchange graph, and then see if there was an augmenting path.

Indeed, when we know all the edges of the exchange graph, we do not need any more (rounds of) queries to figure out if there was an path.[11] If we skipped the $(1-\varepsilon)$-approximation step and just found all the augmenting paths one-by-one we obtain the straightforward $O(n)$-round algorithm.

The difficult part of the algorithm is how to find the $(1-\varepsilon)$-approximation in sublinear number of rounds (even when $1/\varepsilon$ is polynomially large). To do this, we would need to find many augmenting paths simultaneously, and indeed this is our strategy. Our main result of this section is this approximation algorithm which we summarize in Theorem D.4.1 below.

Theorem D.4.1 (Sublinear-round $(1-\varepsilon)$-approximation)**.** *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of size $|S| \geq (1-\varepsilon)r$, where r is the size of the largest common independent set, using either*

- *$O(\sqrt{n}/\varepsilon)$ rounds of polynomially many rank-queries, or*
- *$O(n^{3/4}/\varepsilon)$ rounds of polynomially many independence-queries.*

Exact algorithm. By an appropriate choice of ε ($\varepsilon = n^{-1/4}$ for rank-oracle and $\varepsilon = n^{-1/8}$ for independence-oracle), together with our discussion above, the main result (Theorem D.1.1, restated below) of the paper—the sublinear-round exact algorithm—follows immediately from Theorem D.4.1. The remainder of this paper will go towards proving Theorem D.4.1, i.e. the $(1-\varepsilon)$-approximation algorithm.

Theorem D.1.1 (Sublinear-round Matroid Intersection)**.** *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(n^{3/4})$ rounds of polynomially many rank-queries, or*
- *$O(n^{7/8})$ rounds of polynomially many independence-queries.*

D.4.1 Blocking Flow

The approximation algorithm maintains a common independent set S and runs in $O(1/\varepsilon)$ phases, where in the i'th phase it eliminates all augmenting paths of length $2i$ by finding a blocking flow, similar to the Hopcroft-Karp's [HK73] bipartite matching algorithm and Dinitz's [Din70] max-flow algorithm. By *blocking flow* we mean a set of compatible shortest augmenting paths after which augmenting along them the (s,t)-distance in the exchange graph has increased. At the end, by Lemma D.2.12, we will have found a common independent set S which is a $(1-\varepsilon)$-approximation, since the shortest augmenting path will have length $O(1/\varepsilon)$.

[11] Note that if we also care about the number of rounds of work of the algorithm (and not just the rounds of *queries*), we can find the augmenting path in the exchange graph in just $\text{poly}(n)$ rounds, as s,t-reachability is well-known to be in NC.

This idea of applying blocking flow algorithms to matroid intersection originates from the algorithm of Cunningham [Cun86], but has since been improved by Chakrabarty-Lee-Sidford-Singla-Wong [CLSSW19] and subsequently Blikstad [Bli21], and it is the framework of these two later algorithms which we will follow.

Remark D.4.2. In the first phase we will eliminate all augmenting paths of length 2. This corresponds exactly to finding a *maximal* common independent set, like we did in Section D.3. In general, we will show that we can implement any phase in the same round-complexity as the first phase, using similar ideas.

Beginning of a phase. In each phase, we consider a layered graph, where we let the *distance-layer* D_i denotes all the elements of distance i from the source node s in the exchange graph $G(S)$. At the beginning of a phase, the algorithm will use a single round (of $O(n^2)$ queries) to find these distance layers: simply query all potential edges of the exchange graph.

Unfortunately, knowing all the edges in the exchange graph is not sufficient to find a blocking flow, since a set of disjoint augmenting paths might not be compatible with each other. The exchange graph $G(S)$ does not capture the full structure of the matroid intersection problem, and this is where the difficulty in obtaining sublinear-round matroid intersection algorithms comes from. There is a need to be able to find many compatible augmenting paths "in parallel".

Augmenting sets. The notion of a collection of *compatible*[12] augmenting paths is captured by *augmenting sets*, as defined in Definition D.2.7. So our goal in a phase is to find a *maximal* augmenting set (see Definition D.2.10), which is what we formally mean by "blocking flow". After augmenting along a maximal augmenting set, Lemma D.2.11 implies that the (s,t)-distance has increased, and we can move on to the next phase.

Our algorithm will follow the framework of [CLSSW19] and [Bli21], which are the state-of-the-art sequential independence-query approximation algorithms. The overall idea can be seen as a generalization of the warm-up *maximal* common independent set algorithm from Section D.3. Instead of working with just a single distance layer in the exchange graph we now have up to $O(1/\varepsilon)$ many layers. Fortunately, layers far apart from each other can be handled relatively well in parallel, and we will see that the final adaptivity of our algorithm to find a blocking flow in a phase will not depend on the number of layers.

We start by, on a high level, summarizing how the algorithm of [CLSSW19; Bli21] implements a phase in two stages. Our main result is how we can implement this algorithm efficiently in a parallel.

[12]That is they can all be augmented along simultaneously without breaking independence in either matroid.

1. The first stage keeps track of a *partial* augmenting set (Definition D.2.8) which it keeps *refining* by a series of operations on adjacent distance layers in the exchange graph, to make it closer to a *maximal* augmenting set.

2. When we are "close enough" to a maximal augmenting set, the progress we make in the first stage slows down. Then we fall back to the second stage in which we find the relatively few remaining augmenting paths individually one at a time.

D.4.2 First Stage: Refining

The basic refining ideas and procedures in this section are the same as in [CLSSW19; Bli21]; our contribution is to show how they can be implemented in a parallel fashion.

Say we are in the phase where the (s,t)-distance is $\ell+1$, that is we have ℓ layers in our exchange graph. The algorithm keeps track of a partial augmenting set $\Phi_\ell = (B_1, A_2, B_3, \ldots, A_{\ell-1}, B_\ell)$ (see Definition D.2.8), which it makes local improvements to, called *refining*. Essentially Φ_ℓ looks like a stair-case: B_{k+1} is a set which can be "matched" to some subset of the previous layer A_k; and similarly A_{i+1} can be "matched" to a subset of B_i. As long as Φ_ℓ is "far" from being a maximal augmenting set, the refinement procedures make significant progress. When Φ_ℓ becomes "close" to being a maximal augmenting set we move on to the second stage.

We maintain three types of elements in each layer D_k in the exchange graph.

Selected. Sets A_k and B_k form the partial augmenting set Φ_ℓ.

Removed. Sets R_k contain the discarded elements which we have deemed useless.

Fresh. Sets F_k contain the elements which are neither selected nor removed.

All elements are initially fresh, and for convenience we also define "imaginary" empty boundary layers $D_0 = D_{\ell+1} = \emptyset$, with corresponding sets $A_0, R_0, F_0, A_{\ell+1}$, $R_{\ell+1}, F_{\ell+1}$. Note that (A_k, R_k, F_k) forms a partition of $D_k \subseteq S$ when k is even, and that (B_k, R_k, F_k) partitions $D_k \subseteq V \setminus S$ when k is odd.

The idea of the refinement procedures is to make some local improvements to adjacent distance-layers. While doing this, we make sure that elements only change their types from *fresh* $\to$ *selected* $\to$ *removed*, but never in the other direction. In order to formalize that the removed elements are actually useless, we maintain the following *phase invariants*.

Definition D.4.3 (Phase Invariants, from [CLSSW19, Section 6.3.2])**.** The *phase invariants* are:

(a-b) $\Phi_\ell = (B_1, A_2, B_3, \ldots, A_{\ell-1}, B_\ell)$ forms a partial augmenting set.[13]

[13]The naming of this invariant as (a-b) is to be consistent with [CLSSW19] where this condition is split up into two separate items (a) and (b).

(c) For all even k, $\text{rank}_1(W - R_k) = \text{rank}_1(W) - |R_k|$ where $W = S - A_k + (D_{k+1} - R_{k+1})$.[14]

(d) For all odd k, $\text{rank}_2(W + R_k) = \text{rank}_2(W)$ where $W = S - (D_{k+1} - R_{k+1}) + B_k$.

Remark D.4.4. Invariant (c) and (d) essentially says that if R_{k+1} is useless, then so is R_k, for both even and odd layers, and thus, by induction, all removed elements are indeed useless. For example, (d) says that any element $x \in R_k$ does not increase the rank, even if we take away all non-useless elements $(D_{k+1} - R_{k+1})$ in the next layer. Hence such an x cannot be "matched" to any non-useless element in the next layer, so it is safe to discard it, since we will never be able to add it to B_k while maintaining that $(\ldots, B_k, A_{k+1}, \ldots)$ form an partial augmenting set.

Refining Locally

We now present the basic refinement procedures from [CLSSW19], which are operations on two neighboring layers. We note that [Bli21] improves upon the algorithm of [CLSSW19] (in the sequential setting) by considering refinement operations on **three** consecutive layers instead. Unfortunately, the three-layer refinement procedures of [Bli21] does not seem to work efficiently in the parallel setting.

Intuitively, for an even k, `RefineAB`(k) tries to extend B_{k+1} as much as possible while it still can be "matched" from A_k in the previous layer (i.e. while $S - A_k + B_{k+1} \in \mathcal{I}_1$). After this, if $|A_k| > |B_{k+1}|$, we can remove elements from A_k and argue that they are useless (if they were useful, then it should have been possible to "match" them to something more in the next layer, but this is not the case since B_{k+1} could not be extended more). So `RefineAB`(k) extends B_{k+1} and shrinks A_k so that they are the same size. Doing so, $|A_k^{old}| - |B_{k+1}^{old}|$ elements have changed types, and this crucial observation is what allows us to measure progress. For an odd k, `RefineBA`(k) works very similarly, but now between the consecutive layers (B_k, A_{k+1}).

Algorithm D.2: `RefineAB`(k) for even k (called `Refine1` in [CLSSW19, Algorithm 9])

Data: A_k, B_{k+1}, F_{k+1}, S
Result: Updated A_k, B_{k+1}, F_{k+1}, and R_k
Find a maximal $B \subseteq F_{k+1}$ such that $S - A_k + B_{k+1} + B \in \mathcal{I}_1$
$B_{k+1} \leftarrow B_{k+1} + B$
$F_{k+1} \leftarrow F_{k+1} - B$
Find a maximal $A \subseteq A_k$ such that $S - A_k + B_{k+1} + A \in \mathcal{I}_1$
$A_k \leftarrow A_k - A$
$R_k \leftarrow R_k + A$

[14] This invariant differs from [CLSSW19], where it was written in the following equivalent form: For $1 \le k \le \ell/2$, for any $X \subseteq B_{2k+1} + F_{2k+1} = D_{2k+1} - R_{2k+1}$, if $S - (A_{2k} + R_{2k}) + X \in \mathcal{I}_1$ then $S - A_{2k} + X \in \mathcal{I}_1$.

Algorithm D.3: `RefineBA`(k) for odd k (called `Refine2` in [CLSSW19, Algorithm 10])

Data: $B_k, A_{k+1}, F_{k+1}, S, D_{k+1}, R_{k+1}$
Result: Updated B_k, A_{k+1}, F_k, and R_k
Find a maximal $B \subseteq B_k$ such that $S - (D_{k+1} - R_{k+1}) + B \in \mathcal{I}_2$
$R_k \leftarrow R_k + B_k - B$
$B_k \leftarrow B$
Find a maximal $A \subseteq F_{k+1}$ such that $S - (D_{k+1} - R_{k+1}) + B_k + A \in \mathcal{I}_2$
$A_{k+1} \leftarrow A_{k+1} + F_k - A$
$F_k \leftarrow A$

Remark D.4.5. When we are in the first phase, that is when there is only a single layer between s and t in the graph, running `RefineAB`(0) and `RefineBA`(1) corresponds to our warm-up algorithm to find a *maximal* common independent set from Section D.3. In particular `RefineAB`(0) finds a maximal B_1 such that $S + B_1 \in \mathcal{I}_1$, and `RefineAB`$(1)$ shrinks B_1 such that $S + B_1 \in \mathcal{I}_2$ too.

Lemma D.4.6. *`RefineAB` and `RefineBA` can each be implemented in either:*

- $O(1)$ *rounds of polynomially many rank-queries, or*
- $O(\sqrt{n})$ *rounds of polynomially many independence-queries.*

Proof. The refine procedures only need to find a maximal independent set (for a single matroid) twice, so we can apply Lemma D.3.1. □

The following properties are proven in [CLSSW19].

Lemma D.4.7 (from [CLSSW19, Lemmas 40-42])**.** *Both `RefineAB` and `RefineBA` preserve the phase invariants. Also: after `RefineAB`(k) is run, we have $|A_k| = |B_{k+1}|$ (unless $k = 0$). After `RefineBA`(k) is run, we have $|B_k| = |A_{k+1}|$ (unless $k = \ell$).*

Observation D.4.8. *Lemma D.4.7 can be used to messure progress. In particular, after running `RefineAB`(k), $|A_k| = |B_{k+1}|$, so a total of $|A_k^{old}| - |B_{k+1}^{old}|$ elements must have changed types ($x \in A_k^{old}$ might have been* removed*, while a $x \in F_{k+1}^{old}$ might have been* selected*). Similarly `RefineBA`(k) will change types of $|B_k^{old}| - |A_{k+1}^{old}|$ elements. Note that each element can only change type at most twice (from* fresh *to* selected *to* removed*), so this observation can be used to measure progress.*

Refining Globally

In the sequential algorithms of [CLSSW19; Bli21], a refinement pass consists of running `RefineAB`(k) and `RefineAB`(k) for all k in sequence. However, in the parallel setting we can do better. Since `RefineAB` and `RefineBA` only change things locally

in two adjacent layers, we observe that we can perform several of these refinement operations in parallel.

Algorithm D.4: `Refine()`

1 In parallel, run `RefineAB`(k) for all even $0 \leq k \leq \ell$.
2 In parallel, run `RefineBA`(k) for all odd $0 \leq k \leq \ell$.

Lemma D.4.9. *`Refine` can be implemented in either:*

- $O(1)$ *rounds of polynomially many rank-queries, or*
- $O(\sqrt{n})$ *rounds of polynomially many independence-queries.*

The following Lemma D.4.10 will be useful to bound the number of `Refine` calls needed in our final algorithm, and is similar to [CLSSW19, Corollary 43].

Lemma D.4.10. *Suppose that $|B_k| = |A_{k+1}|$ for all odd $1 \leq k \leq \ell - 1$ and that $S + B_\ell \in \mathcal{I}_2$, before `Refine` is run. After `Refine` is run we have:*

(i) $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$*, still.*

(ii) $S + B_\ell \in \mathcal{I}_2$*, still.*

(iii) $|B'_1| - |B_\ell|$ *elements have changed their type, where* B'_1 *is any maximal subset of* $(D_1 - R_1)$ *such that* $S + B'_1 \in \mathcal{I}_1$*.*

Proof. Property (i) is true, since it is true just after we run `RefineBA`(k), by Lemma D.4.10, and this is what is done in the last step of `Refine`. Similarly property (ii) is true, since `RefineBA`(ℓ) ensures this when "shrinking" B_ℓ (see how B is picked in Algorithm D.3 line 1).

What remains is to prove property (iii). Let $(B_1^{old}, A_2^{old}, \ldots)$ be the sets before `Refine` is run. In the first line of Algorithm D.4, we run `RefineAB`(2), `RefineAB`(4), `RefineAB`(6), ..., which according to Lemma D.4.7 and Observation D.4.8, has incurred a total of $\sum |A_k^{old}| - |B_{k+1}^{old}| = |B_1^{old}| - |B_\ell^{old}|$ type-changes (the sum telescopes since we assume $|B_{k-1}^{old}| = |A_k^{old}|$).

Also note that we run `RefineAB`(0), which extends B_1 until it is a maximal subset of $D_1 \setminus R_1$ such that $S + B_1 \in \mathcal{I}_1$ (line 1 of Algorithm D.2). This means that an additional $|B'_1| - |B_1^{old}|$ elements have changed their type—from *fresh* to *selected*—in the first layer, where B'_1 is the value of B_1 we get after running `RefineAB`(0). Note that this B'_1 is a maximal subset of $(D_1 - R_1)$ such that $S + B'_1 \in \mathcal{I}_1$ (see line 1 of Algorithm D.2).

Hence, in the first line of Algorithm D.4, $|B'_1| - |B_\ell|$ types have changed. We might additionally change types of more elements when running the second line of Algorithm D.4. □

Remark D.4.11. We measure progress in terms of Lemma D.4.10. Since each element can only change types twice (from *fresh* $\mapsto$ *selected* $\mapsto$ *removed*), there will be in total $O(n)$ type-changes. If we just run `Refine`, we might need to do so $O(n)$ times (in the case when $|B_1'| - |B_\ell|$ is constant), so this is not good enough to obtain a sublinear round parallel algorithm. Like we did in the easier case of *maximal* common independent set, we must swap to a different strategy when the progress of refining stagnates, i.e. when $|B_1'| - |B_\ell|$ is relatively small.

D.4.3 Second Stage: Finding the Remaining Augmenting Paths

When $|B_1'| - |B_{\ell+1}|$ is relatively small, we fall back to finding a special kind of augmenting paths one-by-one. We will show that we only need to find $|B_1'| - |B_{\ell+1}|$ many such paths before we get stuck and have found our *maximal* augmenting set (i.e. the desired blocking flow). These special kind of augmenting paths we consider are essentially augmenting paths in the exchange graph *with respect to our partial augmenting set*. They were first introduced by [Bli21], and are called *valid paths*.

Definition D.4.12 (Valid path, from [Bli21, Definition 31])**.** A sequence of elements $(b_i, a_{i+1}, b_{i+2}, \ldots, a_{\ell-1}, b_\ell, t)$ is a *valid path* (w.r.t. our partial augmenting set) *starting at* b_i if for all $k \geq i$:

(a) $a_k \in F_k$ for even k and $b_k \in F_k$ for odd k.

(b) $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$.

(c) $S + B_\ell + b_\ell \in \mathcal{I}_2$.

(d) $S - A_{k+1} + B_k - a_{k+1} + b_k \in \mathcal{I}_2$ for odd k.

(e) $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$ for even k.

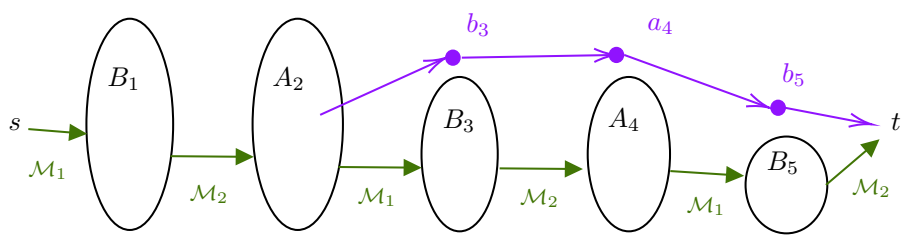


Figure D.1: An example of a valid path (b_3, a_4, b_5, t) starting at b_3.

Remark D.4.13. Compare the definition of valid paths to the edges in the exchange graph from Definition D.2.5. Essentially, items (b-e) corresponds to edges of the exchange graph $G(S + B_1 - A_2 + B_3 - A_4 + \cdots + B_\ell)$ of S *after* augmenting along our partial augmenting set. Note also that item (b) can only hold when $|A_{i-1}| > |B_i|$ (or, when $i = 1$ and $A_0 = \emptyset$ is an "imaginary" boundary set).

Lemma D.4.14 (Augmenting along a valid path [Bli21, Lemma 33])**.** *Suppose that* $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$ *and that* $S + B_\ell \in \mathcal{I}_2$.[15] *If* $(b_i, a_{i+1}, b_{i+2}, \ldots, b_\ell, t)$ *is a valid path starting at* b_i*, then*

$$(B_1, A_2, \ldots, B_{i-2}, A_{i-1},\ B_i + b_i, A_{i+1} + a_{i+1}, \ldots, B_\ell + b_\ell)$$

is a partial augmenting set satisfying the phase invariants and with $S + B_\ell + b_\ell \in \mathcal{I}_2$.

Proof sketch. It is easy to verify that all the properties in the definition of a partial augmenting set are still satisfied after the augmentation. Moreover, the phase invariants are also true, since the sets B_k and A_k are only extended by the augmentation (so an element deemed useless before remains useless). □

Lemma D.4.15. *We can find a valid path, if one exists, in a single round of* $O(n^2)$ *queries.*

Proof. Finding a valid path in a single round of queries is not very different from finding a normal augmenting path. In a single round we query all potential "directed edges", that is all potential (a_k, b_{k+1}) or (b_k, a_{k+1}) pairs satisfying the items of Definition D.4.12 (valid paths). Then we can combine these edges to form a valid path, or else determine that no valid path exist. □

Remark D.4.16. After augmenting along a valid path, $|B_\ell|$ increases by one. Let B_1' be any any maximal subset of $(D_1 - R_1)$ such that $S + B_1' \in \mathcal{I}_1$. Note that we always know that $|B_1'| \geq |B_1| \geq |B_\ell|$. Hence, we need to only find at most $|B_1'| - |B_\ell|$ many valid paths to augment along after we finished the first stage.

We also need the following lemma saying that if, for an element $x \in F_k$, there is no "partial" valid path $(x, \ldots, a_{\ell-1}, b_\ell, t)$ (satisfying all items of a valid path, except maybe item (b) of Definition D.4.12), then it is safe to delete x. We prove this by showing that if x has no "out-edges" (of the form of items (c-e) in Definition D.4.12), then it can be removed.

Lemma D.4.17. *Suppose that* $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$ *and that* $S + B_\ell \in \mathcal{I}_2$*. Then:*

- *If* $b_\ell \in F_k$ *is such that* $S + B_\ell + b_\ell \notin \mathcal{I}_2$*, we can safely remove it.*
- *For a given* $b_k \in F_k$*, if there exist no* $a_{k+1} \in F_{k+1}$ *such that* $S - A_{k+1} + B_k - a_{k+1} + b_k \in \mathcal{I}_2$*, then it is safe to remove* b_k*.*
- *For a given* $a_k \in F_k$*, if there exist no* $b_{k+1} \in F_{k+1}$ *such that* $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$*, then it is safe to remove* a_k*.*

[15] These conditions are actually redundant, since they are covered by items (d) and (c) in the definition of the valid paths. However, they make the intuition slightly easier, and our algorithm maintains them.

Proof sketch. We must argue that the phase invariants are preserved when the elements are removed. It is straightforward to verify that phase invariants (c) and (d) hold in all these three cases. As a black-box intuition, we can imagine temporarily selecting the element x we want to remove, and then running either `RefineAB` or `RefineBA` and note that this procedure can immediately remove x again (and the refine-procedures preserves the invariants). □

D.4.4 Combining the Stages

Now we present the full algorithm of a phase, whose goal is to find a maximal augmenting set, that is a "blocking flow". Pseudo-code can be found in Algorithm D.5, which is parametrized by a cut-off threshold p (which will be different for rank- and independence-query) for when to move from the first to second stage.

Remark D.4.18. After the two stages, we will have some partial augmenting set $(B_1, A_2, \ldots, B_\ell)$ such that there are no more valid paths. However, it is not yet an actual augmenting set, for instance it can be the case that $|A_k| > |B_{k+1}|$ for some k. Still, we can argue that $(B_1, A_2, \ldots, B_\ell)$ contains some *maximal augmenting set* $(\tilde{B}_1, \tilde{A}_2, \ldots, \tilde{B}_\ell)$ with $\tilde{B}_\ell = B_\ell$. So we will need a short extra clean-up step to reduce our partial augmenting set to such a maximal augmenting set.

Note that it is possible to show that we actually can directly augment along our partial augmenting set $(B_1, A_2, \ldots, B_\ell)$ which the algorithm finds (this relies on the extra properties that $|B_k| = |A_{k+1}|$ and $S + B_\ell \in \mathcal{I}_2$). That is $S' = S + B_1 - A_2 + B_2 - \cdots + B_\ell \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a common independent set. Additionally $|S'| = |S| + |B_\ell|$, so we have increased the size of S as much as we would have if we found the the maximal augmenting set $(\tilde{B}_1, \tilde{A}_2, \ldots, \tilde{B}_\ell)$ instead. However, there is a critical problem with this approach: *there can be short augmenting paths in* $G(S')$. This means that such an approach will have failed to eliminate all (s,t)-paths of length $\leq \ell$. Hence the clean-up step is actually necessary.

Correctness. In the beginning of Algorithm D.5 the following hold: (i) the phase invariants (Definition D.4.3); (ii) $|B_k| = |A_{k+1}|$ for all odd $1 \leq k \leq \ell - 1$; and (iii) $S + B_\ell \in \mathcal{I}_2$.

In the first stage, whenever we call `Refine`, the above properties (i-iii) are all preserved according to Lemma D.4.10. Similarly, in the second stage, whenever we augment along a valid path, the above properties (i-iii) are also preserved, by Lemma D.4.14.

What remains to be shown is that after the clean-up phase, $\Phi = (B_1, A_2, \ldots, B_\ell)$ is a maximal augmenting set. We prove this by showing that these refine calls cannot *select* any new elements, that is they do not add any elements to the sets B_k or A_k. If we show this, then we know that $|B_\ell| = |A_{\ell-1}| = \ldots = |B_1|$ after all these refine calls, as `RefineAB`$(\ell - 1)$ reduced $|A_{\ell-1}|$ to match $|B_\ell|$; `RefineBA`$(\ell - 2)$ reduces $|B_{\ell-2}|$ to match $|A_{\ell-1}|$; etc.

Algorithm D.5: Implementation of phase $(\ell+1)/2$.

Data: S, ℓ
Result: Updated set S after phase $(\ell+1)/2$

1 In parallel query all potential edges of the exchange graph $G(S)$ (see Definition D.2.5)
2 Find the distance layers $D_1, D_2, \ldots D_\ell$
3 Initialize $B_k = \emptyset$, $A_k = \emptyset$, $R_k = \emptyset$ and $F_k = D_k$ for all k
4 **while** *true* **do**
    `// Stage 1`
5 Find a maximal $B' \subseteq D_1 - R_1$ such that $S + B' \in \mathcal{I}_1$ (using Lemma D.3.1)
6 **if** $|B'| - |B_\ell| \leq p$ **then**
7 **break**
8 Call `Refine()` (Algorithm D.4)
9 **while** *true* **do**
    `// Stage 2`
10 In a single round, find a valid path if one exists (Lemma D.4.15)
11 **if** *no valid path exists* **then**
12 **break**
13 Augment the partial augmenting set along the found valid path (Lemma D.4.14)
  `// Clean-up`
14 Remove all elements which do not have any partial valid path from them (see Lemma D.4.17)
15 Sequentially, call `RefineBA`(ℓ), `RefineAB`$(\ell-1)$, `RefineBA`$(\ell-2)$, $\ldots$ `RefineAB`(0)
16 Augment along the *maximal augmenting set* $\Phi = (B_1, A_2, \ldots, B_\ell)$: update $S \leftarrow S + B_1 - A_2 + B_3 + \cdots + B_\ell$

To argue that `RefineAB`(k) does not select any new elements, we note that if it added b_k to B_k, it meant that $S - A_{k+1} + B_k + b_k \in \mathcal{I}_1$. However, since b_k was not removed in line 12 of the algorithm, there must have been a valid path starting from b_k, which is a contradiction. The argument for `RefineBA`(k) is the same. Note that since we only remove elements, no new valid paths can occur.

Now, after $|B_1| = |A_2| = \ldots = |B_\ell|$, $(B_1, A_2, \ldots, B_\ell)$ forms a *maximal* augmenting set. If it did not, there must have been some path $(b_1, a_2, \ldots, b_\ell)$ which we can add to it, but this is impossible, since this path would have been a valid path (starting at b_1).

Rounds of adaptivity. The first stage of Algorithm D.5 runs in $O(n/p \cdot \mathcal{T}_{\texttt{Refine}})$ rounds if $\mathcal{T}_{\texttt{Refine}}$ is the number of rounds needed to run `Refine`() once (Lemma D.4.9 gives $\mathcal{T}^{\text{rank}}_{\texttt{Refine}} = O(1)$ and $\mathcal{T}^{\text{indep}}_{\texttt{Refine}} = O(\sqrt{n})$). This is since each time we run refine we will have at least p type-changes (Lemma D.4.10), and in total each of the n elements can change types at most twice.

The second stage of Algorithm D.5 runs in $O(p)$ rounds, both for independence and rank-oracle, by Lemma D.4.15. Picking p optimally gives: $O(\sqrt{n})$ rounds of rank-queries (with $p = \sqrt{n}$) or $O(n^{3/4})$ rounds of independence-queries (with $p = n^{3/4}$) for the first and second stages combined.

Finally the clean-up stage runs, sequentially, with $O(\ell)$ refinement operations. This takes $O(\ell)$ rounds of rank-queries or $O(\ell\sqrt{n})$ rounds of independence queries, so this depends on the number of layers. However, we argue that we can ignore this term for the interesting range of ℓ. This is because when ℓ is too large ($> \sqrt{n}$ for rank-queries and $> n^{1/4}$ for independence-queries), we know by Lemma D.2.12 that there are only $O(1/\ell)$ many augmenting paths left in total, and we can instead find them one-by-one in at most $O(\sqrt{n})$ rounds for rank-queries or $O(n^{3/4})$ rounds for independence queries.

Concluding, we have argued that we can implement a blocking-flow phase in $O(\sqrt{n})$ rounds of rank-queries or $O(n^{3/4})$ rounds of independence-queries.

Approximation algorithm. Running Algorithm D.5 for $O(1/\varepsilon)$ phases eliminates all paths in the exchange graph of length $O(1/\varepsilon)$ (Lemma D.2.11), so by Lemma D.2.12 we know that the common independent set S we end up with is a $(1-\varepsilon)$-approximation. The adaptivity is thus $O(\sqrt{n}/\varepsilon)$ rounds of rank-queries or $O(n^{3/4}/\varepsilon)$ rounds of independence-queries. Hence we have shown a $(1-\varepsilon)$-approximation algorithm using $O(\sqrt{n}/\varepsilon)$ rounds of (polynomially many) rank-queries or $O(n^{3/4}/\varepsilon)$ rounds of (polynomially many) independence-queries, which proves Theorem D.4.1.

Bibliography

[AD71] Martin Aigner and Thomas A. Dowling. "Matching Theory for Combinatorial Geometries". In: *Transactions of the American Mathematical Society* 158.1 (1971), pp. 231–245 (cit. on p. 245).

[BBMN21] Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Breaking the quadratic barrier for matroid intersection". In: *STOC*. ACM, 2021, pp. 421–432. DOI: 10.1145/3406325.3451092 (cit. on pp. 245, 249).

[Bli21] Joakim Blikstad. "Breaking O(nr) for Matroid Intersection". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 31:1–31:17. DOI: 10.4230/LIPIcs.ICALP.2021.31 (cit. on pp. 245, 247–249, 255–258, 260, 261).

[CCK21] Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. "A Polynomial Lower Bound on the Number of Rounds for Parallel Submodular Function Minimization and Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2021, pp. 37–48. DOI: 10.1109/FOCS52979.2021.00013 (cit. on pp. 246, 247).

[CLSSW19] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. "Faster Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2019, pp. 1146–1168. DOI: 10.1109/FOCS.2019.00072 (cit. on pp. 245, 247–250, 255–259).

[Cun86] William H. Cunningham. "Improved Bounds for Matroid Partition and Intersection Algorithms". In: *SIAM J. Comput.* 15.4 (1986), pp. 948–957. DOI: 10.1137/0215066 (cit. on pp. 245, 247, 250, 255).

[Din70] Efim A Dinic. "Algorithm for solution of a problem of maximum flow in networks with power estimation". In: *Soviet Math. Doklady*. Vol. 11. 1970, pp. 1277–1280 (cit. on pp. 247, 254).

[Edm70] Jack Edmonds. "Submodular functions, matroids, and certain polyhedra". In: *Combinatorial structures and their applications*. Gordon and Breach, 1970, pp. 69–87 (cit. on p. 245).

[Edm79] Jack Edmonds. "Matroid intersection". In: *Annals of discrete Mathematics*. Vol. 4. Elsevier, 1979, pp. 39–49 (cit. on p. 245).

[EDVJ68] Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. "Matroid partition". In: *50 Years of Integer Programming 1958–2008* (1968), p. 199 (cit. on p. 245).

[FGT21] Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. "Bipartite Perfect Matching is in Quasi-NC". In: *SIAM J. Comput.* 50.3 (2021). DOI: 10.1137/16M1097870 (cit. on p. 246).

[GGR22] Sumanta Ghosh, Rohit Gurjar, and Roshan Raj. "A Deterministic Parallel Reduction from Weighted Matroid Intersection Search to Decision". In: *SODA*. SIAM, 2022, pp. 1013–1035. DOI: 10.1137/1.9781611977073.44 (cit. on p. 246).

[GT20] Rohit Gurjar and Thomas Thierauf. "Linear Matroid Intersection is in Quasi-NC". In: *Comput. Complex.* 29.2 (2020), p. 9. DOI: 10.1007/s00037-020-00200-z (cit. on p. 246).

[HK73] 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. DOI: 10.1137/0202019 (cit. on pp. 247, 254).

[KUW85] Richard M. Karp, Eli Upfal, and Avi Wigderson. "The Complexity of Parallel Computation on Matroids". In: *FOCS*. IEEE Computer Society, 1985, pp. 541–550. DOI: 10.1109/SFCS.1985.57 (cit. on pp. 246–248, 251, 252).

[KUW86] Richard M. Karp, Eli Upfal, and Avi Wigderson. "Constructing a perfect matching is in random NC". In: *Comb.* 6.1 (1986), pp. 35–48. DOI: 10.1007/BF02579407 (cit. on p. 246).

[KUW88] Richard M. Karp, Eli Upfal, and Avi Wigderson. "The Complexity of Parallel Search". In: *J. Comput. Syst. Sci.* 36.2 (1988), pp. 225–253. DOI: 10.1016/0022-0000(88)90027-X (cit. on p. 251).

[Law75] Eugene L. Lawler. "Matroid intersection algorithms". In: *Math. Program.* 9.1 (1975), pp. 31–56. DOI: 10.1007/BF01681329 (cit. on p. 245).

[Lov79] László Lovász. "On determinants, matchings, and random algorithms". In: *FCT*. Akademie-Verlag, Berlin, 1979, pp. 565–574 (cit. on p. 246).

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. "A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization". In: *FOCS*. IEEE Computer Society, 2015, pp. 1049–1065. DOI: 10.1109/FOCS.2015.68 (cit. on p. 245).

[Ngu19] Huy L. Nguyen. "A note on Cunningham's algorithm for matroid intersection". In: *CoRR* abs/1904.04129 (2019) (cit. on p. 245).

Paper E

Fast Algorithms via Dynamic-Oracle Matroids

Joakim Blikstad, Sagnik Mukhopadhyay,
Danupon Nanongkai, Ta-Wei Tu

Article published in Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023. [BMNT23]
Full version at `https://arxiv.org/abs/2302.09796`.

Abstract

We initiate the study of matroid problems in a new oracle model called *dynamic oracle*. Our algorithms in this model lead to new bounds for some classic problems, and a "unified" algorithm whose performance matches previous results developed in various papers for various problems. We also show a lower bound that answers some open problems from a few decades ago. Concretely, our results are as follows.

Improved algorithms for matroid union and disjoint spanning trees. We show an algorithm with $\tilde{O}_k(n + r\sqrt{r})$ dynamic-rank-query and time complexities for the matroid union problem over k matroids, where n is the input size, r is the output size, and $\tilde{O}_k$ hides $\text{poly}(k, \log(n))$. This implies the following consequences. **(i)** An improvement over the $\tilde{O}_k(n\sqrt{r})$ bound implied by [Chakrabarty-Lee-Sidford-Singla-Wong FOCS'19] for matroid union in the traditional rank-query model. **(ii)** An $\tilde{O}_k\left(|E| + |V|\sqrt{|V|}\right)$-time algorithm for the k-disjoint spanning tree problem. This is nearly linear for moderately dense input graphs and improves the $\tilde{O}_k\left(|V|\sqrt{|E|}\right)$ bounds of Gabow-Westermann [STOC'88] and Gabow [STOC'91]. Consequently, this gives improved bounds for, e.g., Shannon Switching Game and Graph Irreducibility.

Matroid intersection. We show a matroid intersection algorithm with $\tilde{O}(n\sqrt{r})$ dynamic-rank-query and time complexities. This implies new bounds for some problems (e.g. maximum forest with deadlines) and bounds that match the classic ones obtained in various papers for various problems, e.g. colorful spanning tree [Gabow-Stallmann ICALP'85], graphic matroid intersection [Gabow-Xu FOCS'89], simple job scheduling matroid intersection [Xu-Gabow ISAAC'94], and Hopcroft-Karp combinatorial bipartite matching. More importantly, this is done via a "unified" algorithm in the sense that an improvement over our dynamic-rank-query algorithm would imply improved bounds for *all* the above problems simultaneously.

Lower bounds. We show simple super-linear ($\Omega(n \log n)$) query lower bounds for matroid intersection and union problems in our dynamic-rank-oracle and the traditional independence-query models; the latter improves the previous $\log_2(3)n - o(n)$ bound by Harvey [SODA'08] and answers an open problem raised by, e.g., Welsh [1976] and CLSSW [FOCS'19].

E.1 Introduction

Via reductions to the max-flow and min-cost flow problems, exciting progress has been recently made for many graph problems such as maximum matching, vertex connectivity, directed cut, and Gomory-Hu trees [Mad13; LS14; Mad16; BLNPSSSW20; KLS20; LP20; AKT21b; BLLSSSW21; LNPSY21; AKT21a; AMV21; CLNPSQ21; LPS21; GLP21; AKT22; CLP22; BGJLLPS22; AKLPST22; CKLPGS22; CHLP23]. However, many basic problems still witness no progress since a few decades ago. These problems include k-disjoint spanning trees [GW88; Gab91], colorful spanning tree [GS85], arboricity [Gab95], spanning tree packing [GW88], graphic matroid intersection [GS85; GX89], and simple job scheduling matroid intersection [XG94]. For example, in the *k-disjoint spanning trees problem* [Sch03, Chapter 51], we want to find k edge-disjoint spanning trees in a given input graph $G = (V, E)$. When $k = 1$, this is the spanning tree problem and can be solved in linear time. For higher values of k, the best runtime remains $\tilde{O}(k^{3/2}|V|\sqrt{|E|})$-time algorithm from around 1990 [GW88][1], which is also the best runtime for its applications such as Shannon Switching Game [Gar61] and Graph k-Irreducibility [Whi88; GSS93]. No better runtime was known even for the special case of $k = 2$.

Can we improve the bounds of k-disjoint spanning trees and other problems? More importantly, since it is very unclear if these problems can be reduced to max-flow or min-cost flow[2], *is there an alternative approach to designing fast algorithms for many problems simultaneously?* Fortunately, many of the above problems can be modeled as *matroid problems*, giving hope that solving matroid problems would solve many of these problems in one shot. Unfortunately, this is not true in the traditional model for matroid problems—even the most efficient algorithm possible for a matroid problem does not necessarily give a faster algorithm for any of its special cases. We discuss this more below.

Matroid Problems. A matroid $\mathcal{M}$ is a pair $(U, \mathcal{I})$ where U is a finite set (called the ground set) and $\mathcal{I}$ is a family of subsets of U (called the independent sets) satisfying some constraints (see Definition E.3.1; these constraints are not important in the following discussion). Since $\mathcal{I}$ can be very large, problems on matroid $\mathcal{M}$ are usually modeled with *oracles* that answer *queries*. Given a set $S \subseteq U$, *independence queries* ask if $S \in \mathcal{I}$ and *rank queries* ask for the value of $\max_{I \in \mathcal{I}, I \subseteq S} |I|$. Two

[1]The stated bound was due to Gabow and Westermann [GW88], which was usually referred to as the state of the art (e.g. in [Sch03; BF20; Qua23; HSV21]). Note that Gabow [Gab91] announced an improved bound of $O(k|V|\sqrt{|E| + k|V|\log(|V|)})$ but this bound was later removed from the journal version of the paper. After our paper was accepted to STOC'23, we are aware of the paper by Karger [Kar98] that also studies this problem. The paper claims the runtime of $\tilde{O}(|E| + k^{5/2}|V|^3)$ (via matroid union), but it seems that the technique in the paper may imply $\tilde{O}(|E| + \text{poly}(k)|V|^{3/2})$ runtime when combined with [GW88]. We discuss a relevant concurrent result [Qua23] later in this section.

[2]For example, the best-known number of max-flow calls to decide whether there are k disjoint spanning trees and to find the k spanning trees are $O(n)$ and $O(n^2)$ respectively.

textbook examples of matroid problems are *matroid intersection* and *union*[3] (e.g., [Sch03, Chapters 41-42]). We will also consider the special case of matroid union called *k-fold matroid union*.

Definition E.1.1 (Matroid intersection and (k-fold) matroid union)**.** (I) In matroid intersection, we are given two matroids $(U, \mathcal{I}_1)$ and $(U, \mathcal{I}_2)$ and want to find a set of maximum size in $\mathcal{I}_1 \cap \mathcal{I}_2$. (II) In matroid union, we are given k matroids $(U_1, \mathcal{I}_1), (U_2, \mathcal{I}_2), \ldots, (U_k, \mathcal{I}_k)$, and want to find the set $S_1 \cup S_2 \cup \cdots \cup S_k$, where $S_i \in \mathcal{I}_i$ for every i, of maximum size. (III) Matroid union in the special case where $U_1 = U_2 = \cdots = U_k$ and $\mathcal{I}_1 = \mathcal{I}_2 = \cdots = \mathcal{I}_k$ is called k-fold matroid union.

Notations: Throughout, for problems over matroids $\{(U_i, \mathcal{I}_i)\}_{i=1}^k$, we define $n := \max_i |U_i|$ and $r := \max_i \max_{S \in \mathcal{I}_i} |S|$. □

Matroid problems are powerful abstractions that can model many fundamental problems. For example, the 2-disjoint spanning tree problem can be modeled as a 2-fold matroid union problem:

> Given a graph $G = (V, E)$, let $\mathcal{M} = (U, \mathcal{I})$ be the corresponding *graphic matroids*, i.e. $U = E$ and $S \subseteq E$ is in $\mathcal{I}$ if it is a forest in G. (It is a standard fact that such an $\mathcal{M}$ is a matroid.) The 2-fold matroid union problem with input $\mathcal{M}$ is a problem of finding two forests $F_1 \subseteq E$ and $F_2 \subseteq E$ in G that maximizes $|F_1 \cup F_2|$. This is known as the 2-*forest* problem which clearly generalizes 2-disjoint spanning tree (a 2-forest algorithm will return two disjoint spanning trees in G if they exist).

Observe that this argument can be generalized to modeling the k-disjoint spanning trees problem by k-fold matroid union. Other problems that can be modeled as matroid union (respectively, matroid intersection) include arboricity, spanning tree packing, k-pseudoforest, and mixed k-forest-pseudoforest (respectively, bipartite matching and colorful spanning tree).

The above fact makes matroid problems a *unified* approach for showing that many problems, including those mentioned above, can be solved in polynomial time. This is because (i) the matroid union, intersection, and other problems can be solved in polynomial time and rank/independence queries, and (ii) for most problems queries can be answered in polynomial time. For example, when we model k-disjoint spanning trees as k-fold graphic matroid union like above, the corresponding rank query is: given a set S of edges, find the size of a spanning forest of S. This can be solved in $O(|S|)$ time.

When it comes to more fine-grained time complexities, such as nearly linear and sub-quadratic time, matroid algorithms in the above model are not very helpful. This is because simulating a matroid algorithm in this model causes too much runtime blow-up. For example, even if we can solve k-fold matroid union over $(U, \mathcal{I})$ in *linear* ($O(|U|)$) rank query complexity, it does not necessarily imply that we can

[3] *Matroid union* is also sometimes called *matroid sum*.

solve its special case of 2-disjoint spanning tree any faster. This is because each query about a set S of edges needs at least $O(|S|)$ time even to specify S, which can be as large as the number of edges in the input graph. In other words, even a matroid union algorithm with linear complexities may only imply $O(|E|^2)$ time for solving 2-disjoint spanning trees on graphs $G = (V, E)$. This is also the case for other problems that can be modeled as matroid union and intersection. Because of this, previous works obtained improved bounds by simulating an algorithm for matroid problems and coming up with clever ideas to speed up the simulation for each of these problems one by one (e.g., [GT79; RT85; GS85; GW88; FS89; GX89; Gab91; XG94]). It cannot be guaranteed that recent and future improved algorithms for matroid problems (e.g., [CLSSW19; BBMN21; Bli21]) would imply improved bounds for any of these problems.

Dynamic Oracle. The main conceptual contribution of this paper is an introduction of a new matroid model called *dynamic oracle* and an observation that, using dynamic algorithms, solving a matroid problem efficiently in our model immediately implies efficient algorithms for many problems it can model. In contrast to traditional matroids where a query can be made with an arbitrary set S, our model only allows queries made by slightly modifying the previous queries.[4] More precisely, the dynamic-rank-oracle model, which is the focus of this work, is defined as follows.[5]

Definition E.1.2 (Dynamic-rank-oracle model)**.** For a matroid $\mathcal{M} = (U, \mathcal{I})$, starting from $S_0 = \emptyset$ and $k = 0$, the algorithm can access the oracle via the following three operations.

- INSERT(v, i): Create a new set $S_{k+1} := S_i \cup \{v\}$ and increment k by one.
- DELETE(v, i): Create a new set $S_{k+1} := S_i \setminus \{v\}$ and increment k by one.
- QUERY(i): Return the rank of S_i, i.e., the size of the largest independent subset of S_i.

We say that a matroid algorithm takes t time and dynamic-rank-query complexities if its time complexity and required number of operations are both at most t. □

We emphasize that a query can be obtained from *any* previous query, not just the last one.

Observation E.1.3 (Details in Sections E.7.3 and E.11)**.** *Algorithms for the k-fold matroid union, matroid union, and matroid intersection problems imply algorithms for a number of problems with time complexities shown in Section E.1.*

Proof Idea. As an example, we sketch the proof that if k-fold matroid union can be solved in $T(n, r, k)$ then k-disjoint spanning trees can be found in $T(|E|, |V|, k) \cdot$

[4] The "cost" of a query in our dynamic model is the distance (size of the symmetric difference) from some (not necessarily the last) previous query.

[5] One can also define the dynamic-independence-oracle model where QUERY(i) returns only the independence of S_i.

Table E.1: Examples of implications of dynamic-rank-oracle matroid algorithms. The complexities in the first column are in terms of time and dynamic-rank-query complexities. Notations n, r, and k are as in Definition E.1.1. In the second column, the polylog$n$ factors are hidden in $\tilde{T}$ and subpolynomial factors are hidden in $\hat{T}$. Details are in Sections E.7.3 and E.11.

matroid problems	**special cases**
k-fold matroid union in $T(n, r, k)$	k-forest in $\tilde{T}(\|E\|, \|V\|, k))$ k-pseudoforest in $\tilde{T}(\|E\|, \|V\|, k))$ k-disjoint spanning tree in $\tilde{T}(\|E\|, \|V\|, k)$ (randomized) $\hat{T}(\|E\|, \|V\|, k)$ (deterministic) arboricity in $\tilde{T}(\|E\|, \|V\|, \sqrt{\|E\|}))$ tree packing in $\tilde{T}(\|E\|, \|V\|, \|E\|/\|V\|))$ Shannon Switching Game in $\tilde{T}(\|E\|, \|V\|, 2))$ graph k-irreducibility in $\tilde{T}(\|E\|, \|V\|, k))$
matroid union in $T(n, r, k)$	(f, p)-mixed forest-pseudoforest in $\tilde{T}(\|E\|, \|V\|, f + p))$
matroid intersection in $T(n, r)$	bipartite matching in $\tilde{T}(\|E\|, \|V\|)$ colorful spanning tree in $\tilde{T}(\|E\|, \|V\|)$ graphic matroid intersection in $\tilde{T}(\|E\|, \|V\|)$ simple job scheduling matroid intersection in $\tilde{T}(n, r)$ convex transversal matroid intersection in $\tilde{T}(\|V\|, \mu)$

polylog($|V|$) time. Recall that in the traditional rank-oracle model, the algorithm can ask an oracle for the size of a spanning forest in an arbitrary set of edges S, causing $O(|S|)$ time to simulate. In our dynamic-rank-oracle model, an algorithm needs to modify some set S_i to the desired set S using the INSERT and DELETE operations before asking for the size of a spanning forest in S. We can use a spanning forest data structure to keep track of the size of the spanning forest under edge insertions and deletions. This takes polylog($|V|$) time per operation [KKM13; GKKT15].[6] So, if k-fold matroid union can be solved in $T(n, r, k)$ time and dynamic rank queries, then k-disjoint spanning trees can be solved in $\tilde{O}(T(n, r, k)) = \tilde{O}(T(|E|, |V|, k))$ time, where the equality is because the ground set size is the number of edges ($|U| = |E|$) and the rank r is equal to the size of a spanning forest (thus at most $|V|$).[7] □

Observe that designing efficient algorithms in our dynamic-oracle model is *not*

[6]The dynamic spanning forest algorithms of [KKM13; GKKT15] are randomized and assume the so-called oblivious adversary (as opposed to, e.g., [NS17; Wul17] which work against adaptive adversaries). This is not a problem because we only need to report the size of the spanning forest and not an actual forest. We can also use a deterministic algorithm from [CGLNPS20; NSW17] which requires $|V|^{o(1)}$ time per operation.

[7]Note that we also need a fully-persistent data structure [DSST86; Die89] to maintain the whole change history in our argument.

easier than in the traditional model: a dynamic-oracle matroid algorithm can be simulated in the traditional model within the same time and query complexities. Naturally, the first challenge of the new model is this question: *Can we get matroid algorithms in the new model whose performances match the state-of-the-art algorithms in the traditional model?* Moreover, for the new model to provide a unified approach to solve many problems simultaneously, one can further ask: *Would these new matroid algorithms imply state-of-the-art bounds for many problems?*

Algorithms. In this paper, we provide algorithms in the new model whose complexities not only match those in the traditional model but sometimes even improve them. These lead to new bounds for some problems and, for other problems, a unified algorithm whose performance matches previous results developed in various papers for various problems.

More precisely, the best time and rank-query complexities for matroid intersection on input $(U, \mathcal{I}_1)$ and $(U, \mathcal{I}_2)$ were $\tilde{O}(n\sqrt{r})$ by Chakrabarty, Lee, Sidford, Singla, and Wong [CLSSW19] (improving the previous $\tilde{O}(nr)$ bound based on Cunningham's classic algorithm [Cun86; LSW15; Ngu19]). Due to a known reduction, this implies $\tilde{O}(k^2\sqrt{k}n\sqrt{r})$ bound for k-fold matroid union and matroid union. In this paper, we present algorithms in the dynamic-oracle model that imply improved bounds in the traditional model for k-fold matroid union and matroid union and match the bounds for matroid intersection.

Here, we only state our dynamic-rank-query complexities as they are the main focus of this paper, and for all the applications we have, answering (and maintaining dynamically) independence queries does not seem to be significantly easier. Note that we also obtain dynamic-independence-query algorithms that match the state-of-the-art traditional ones [Bli21] which we defer to Section E.12.

Theorem E.1.4. *(I) k-fold matroid union over input $(U, \mathcal{I})$ can be solved in $\tilde{O}(n + kr\sqrt{\min(n, kr)} + k\min(n, kr))$ time and dynamic rank queries. (II) Matroid union over input $(U_1, \mathcal{I}_1), (U_2, \mathcal{I}_2), \ldots, (U_k, \mathcal{I}_k)$ can be solved in $\tilde{O}\left((n + r\sqrt{r}) \cdot \mathrm{poly}(k)\right)$ time and dynamic rank queries. (III) Matroid intersection over input $(U, \mathcal{I}_1)$ and $(U, \mathcal{I}_2)$ can be solved in $\tilde{O}(n\sqrt{r})$ time and dynamic rank queries.*

Combined with Observation E.1.3, the above theorem immediately implies fast algorithms for many problems. Table E.2 shows some of these problems. One of our highlights is the improved bounds for k-forest and k-disjoint spanning trees. Even for $k = 2$, there was no runtime better than the decades-old $\tilde{O}(k^{3/2}|V|\sqrt{|E|})$ runtime [GW88; Gab91]. Our result improves this to $\tilde{O}(|E| + (k|V|)^{3/2})$. This is nearly linear for dense input graphs and small k. This also implies a faster runtime

[8] For k-forest and its related graph problems in the table, we can assume that $k \leq |V|$, and thus the $k^2 r$ (where $r = \Theta(|V|)$) term in Theorem E.1.4 is dominated by the $(kr)^{3/2}$ term.

[9] Here we use the bound that $\alpha \leq \sqrt{E}$ [Gab95].

[10] Our bound is with respect to the current value of $\omega < 2.37286$ [AW21]. If $\omega = 2$, then our bound becomes $\tilde{O}(n^{2.5}\sqrt{r})$.

Table E.2: Implications of our matroid algorithms stated in Theorem E.1.4 in comparison with previous results. Results marked with a ✓ improve over the previous ones. Results marked with a ✗ are worse than the best time bounds. Other results match the currently best-known algorithms up to poly-logarithmic factors. Details can be found in Section E.11.

problems	our bounds	state-of-the-art results
(Via k-fold matroid union)		
k-forest[8]	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2})$ ✓	$\tilde{O}(k^{3/2}\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
k-pseudoforest	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2})$ ✗	$\lvert E\rvert^{1+o(1)}$ [CKLPGS22]
k-disjoint spanning trees	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2})$ ✓	$\tilde{O}(k^{3/2}\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
arboricity[9]	$\tilde{O}(\lvert E\rvert\lvert V\rvert)$ ✗	$\tilde{O}(\lvert E\rvert^{3/2})$ [Gab95]
tree packing	$\tilde{O}(\lvert E\rvert^{3/2})$	$\tilde{O}(\lvert E\rvert^{3/2})$ [GW88]
Shannon Switching Game	$\tilde{O}(\lvert E\rvert + \lvert V\rvert^{3/2})$ ✓	$\tilde{O}(\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
graph k-irreducibility	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2} + k^2\lvert V\rvert)$ ✓	$\tilde{O}(k^{3/2}\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
(Via matroid union)		
(f,p)-mixed forest-pseudoforest	$\tilde{O}_{f,p}(\lvert E\rvert + \lvert V\rvert\sqrt{\lvert V\rvert})$ ✓	$\tilde{O}((f+p)\lvert V\rvert\sqrt{f\lvert E\rvert})$ [GW88]
(Via matroid intersection)		
bipartite matching (combinatorial)	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$	$O(\lvert E\rvert\sqrt{\lvert V\rvert})$ [HK73]
bipartite matching (continuous)	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ ✗	$\lvert E\rvert^{1+o(1)}$ [CKLPGS22]
graphic matroid intersection	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ [GX89]
simple job scheduling matroid intersection	$\tilde{O}(n\sqrt{r})$	$\tilde{O}(n\sqrt{r})$ [XG94]
convex transversal matroid [EF65] intersection	$\tilde{O}(\lvert V\rvert\sqrt{\mu})$	$\tilde{O}(\lvert V\rvert\sqrt{\mu})$ [XG94]
linear matroid intersection[10]	$\tilde{O}(n^{2.529}\sqrt{r})$ ✗	$\tilde{O}(nr^{\omega-1})$ [Har09]
colorful spanning tree	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ [GS85]
maximum forest with deadlines	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ ✓	(no prior work)

for, e.g., Shannon Switching Game (see [Sha55; GW88]) which is a special case of 2-disjoint spanning trees.

Our matroid intersection algorithm gives a unified approach to achieving time complexities that were previously obtained by various techniques in many papers. Thus, improving this algorithm would imply breakthrough runtimes for many of these problems simultaneously. Moreover, in contrast to the previous approach where matroid algorithms have to be considered for each new problem one by one, our approach has the advantage that it can be easier to derive new bounds. For example, say we are given a graph $G = (V, E)$, where the edge e will stop functioning after day $d(e)$. Every day we can "repair" one functioning edge. Our goal is to make the graph connected in the long run (an edge will work forever once it has been repaired). This is the *maximum forest with deadlines* problem. Formally speaking, the goal is to construct a spanning tree or a forest of the maximum size at the end, by selecting an edge e with $d(e) \geq t$ in the t^{th} round.[11] Our result implies a runtime of $\tilde{O}(|E|\sqrt{|V|})$ for this problem. The runtime holds even for the harder case where each edge is also associated with an *arrival time* (edges cannot be selected before they arrive).

[11] It is tempting to believe that we can use a greedy algorithm where we always select an edge e with the smallest $d(e)$ to the solution. The following example shows why this does not work: There are three vertices $V = \{a, b, c, d\}$. Edges e_1 and e_2 between a and b have $d(e_1) = 1$ and $d(e_2) = 3$. Edges $e_3 = (b, c)$ and $e_4 = (c, d)$ have $d(e_3) = d(e_4) = 2$.

We also list some problems where our bounds cannot match the best bounds in Table E.2. Improving our matroid algorithms to match these bounds is a very interesting open problem. A particularly interesting case is the maximum bipartite matching problem. Our dynamic-oracle matroid intersection algorithm implies a runtime that matches the runtime from the best combinatorial algorithm of Hopcroft and Karp [HK73] which has been recently improved via continuous optimization techniques (e.g., [Mad13; Mad16; CMSV17; AMV20; BLNPSSSW20; BLLSSSW21; CKLPGS22]).[12] There are barriers to using continuous optimization even to solve some special cases of matroid intersection (e.g. colorful spanning tree's linear program requires exponentially many constraints). Thus, improving our matroid intersection algorithm requires either a new way to use continuous optimization techniques or a breakthrough idea in designing combinatorial algorithms that would improve the Hopcroft-Karp algorithm.

Lower Bounds. Another advantage of our dynamic-oracle matroid model is that it can be easier to prove lower bounds than the traditional model. As a showcase, we show a simple super-linear rank-query lower bound in our new model. In fact, our argument also implies the first super-linear independence-query lower bound in the traditional model. The latter result might be of independent interest.

Theorem E.1.5. *(I) Any deterministic algorithms require $\Omega(n \log n)$ dynamic rank queries to solve the matroid union and matroid intersection problems. (II) Any deterministic algorithms require $\Omega(n \log n)$ (traditional) independence queries to solve the matroid union and matroid intersection problems.*

Our first lower bound suggests that the dynamic-oracle model might at best give nearly linear (and not linear) time algorithms. Prior to this paper, only a $\log_2(3)n - o(n)$ independence-query lower bound for deterministic algorithms was known for (traditional) independence queries, due to Harvey [Har08].[13] Our lower bound in the traditional model improves this decade-old bound. Moreover, showing super-linear independence-query lower bounds in the traditional model for matroid intersection is a long-standing open problem considered since 1976 (e.g. [Wel76; CLSSW19]).[14] Our lower bound in the traditional model answers this open problem for deterministic algorithms. The case of randomized algorithms would be resolved too if an $\omega(|V|)$ lower bound was proved for the communication complexity for computing connectivity of an input graph $G = (V, E)$. (It was conjectured to be $\Omega(n \log n)$ in [AEGLMN22].)

[12] The term "combinatorial" is vague and varies in different contexts. Here, an algorithm is "combinatorial" if it does not use any of the continuous optimization techniques such as interior-point methods (IPMs).

[13] To the best of our knowledge, this lower bound does not hold for rank queries.

[14] As noted by Harvey, Welsh asked about the number of queries needed to solve the matroid partition problem, which is equivalent to matroid union and intersection.

Independent Work. Concurrently and independently to our work, [Qua23] also studied the k-fold matroid union and related problems and obtained a similar running time of $\tilde{O}(\min(n, kr)^{3/2})$ to ours in the *traditional independence-oracle model*. By specializing the algorithm to graphic matroids, Quanrud also obtained an $\tilde{O}(\min(|E|, k|V|)^{3/2})$ algorithm for k-disjoint spanning tree. The techniques used in these two works are different, however, and our main contribution remains the introduction of *dynamic* oracles and efficient matroid algorithms in this model.

E.1.1 Techniques

In this section we briefly discuss our technical contributions. For a more in-depth overview of our algorithms, see the technical overview (Section E.2).

Exchange Graph & Blocking Flow. Our algorithms and lower bounds are based on the notion of finding *augmenting paths* in the *exchange graph*, due to [Edm70; Law75; AD71]. Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the exchange graph $G(S)$ is a directed graph where finding an (s, t)-path corresponds to increasing the size of S by one. Starting with the work of Cunningham [Cun86], modern matroid intersection algorithms (including the state-of-the-art [CLSSW19; Bli21]) are based on a "Blocking Flow" idea inspired by the Hopcroft-Karp's [HK73] bipartite matching and Dinic's [Din70] max-flow algorithms.

Matroid Intersection with Dynamic Oracle. Our matroid intersection algorithms are implementations of the state-of-the-art $\tilde{O}(n\sqrt{r})$ rank-query algorithm of [CLSSW19] and the $\tilde{O}(nr^{3/4})$ independence-query algorithm of [Bli21]. Our contribution here is to show that versions of them can be implemented also in the *dynamic*-oracle model.

These algorithms explore the exchange graph efficiently in the classic non-dynamic models by performing binary searches with the oracle queries to find useful edges. However, such a binary search is very expensive in the dynamic-oracle model (as the queries differ by a lot): a single such binary search might cost up to $O(n)$ in the dynamic-oracle model instead of just $O(\log n)$.

Our contribution is to design a binary-tree data structure that supports finding these useful edges efficiently also in the dynamic-oracle model. Note that after each augmentation the underlying exchange graph changes, so the data structure must also support these dynamic updates efficiently. Some updates can just be propagated up the tree, while others we handle by batching them and rebuilding the tree periodically. We also rely on a structural result "Augmenting Sets" by [CLSSW19] which states that the updates to the exchange graph are local, which helps us reduce the number of updates we need to make to our data structure, and achieve the final time bound.

Matroid Union with Dynamic Oracle. Our $\tilde{O}(n + r\sqrt{r})$ matroid union algorithm with dynamic rank oracle is based on our $\tilde{O}(n\sqrt{r})$ matroid intersection algorithm (indeed, matroid union is a special case of matroid intersection). We are able to obtain a more efficient algorithm by taking advantage of the additional structure of the exchange graph in the case of matroid union. The main idea is to run the blocking flow algorithm only on a dynamically-changing subgraph of size $\Theta(r)$, instead of on the full exchange graph of size $\Theta(n)$.

A crucial observation is that all but $O(r)$ elements will be directly connected to the source vertex s. To "sparsify" this first layer in the breadth-first-search tree, we argue that one only needs to consider a basis of it (this basis will have size at most r as opposed to n). After an augmentation, this first layer changes, so we design a *dynamic algorithm to maintain a basis of a matroid*[15], with $\tilde{O}(\sqrt{r})$ update time and $O(n)$ pre-computation. Our algorithm to maintain this basis dynamically is inspired by the dynamic minimum spanning tree algorithm of [Fre85] ($O(\sqrt{|E|})$ update time), in combination with the sparsification trick of [EGIN97] ($\tilde{O}(\sqrt{|V|})$ update time). We believe that our dynamic algorithm to maintain a (min-weight) basis of a matroid might also be of independent interest.

Lower Bounds. Our super-linear $\Omega(n \log n)$ query lower bound comes from studying the communication complexity of matroid intersection. The matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ are given to two parties Alice and Bob respectively and they are asked to solve the matroid intersection problem using as few bits of communication between them. We show that even if Alice and Bob know some common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, they need to communicate $\Omega(n \log n)$ bits to see if S is optimal. Essentially, they need to determine if there is an augmenting path in the exchange graph. Using a class of matroids called *gammoids* (see e.g. [Per68; Mas72]), we show a reduction from the (s, t)-connectivity problem which has a deterministic $\Omega(n \log n)$ communication lower bound [HMT88].

E.1.2 Organization

The rest of the paper is organized as follows. We first give a high-level overview of how we obtain our algorithms in Section E.2. In Section E.3, we provide the necessary preliminaries. We then construct the binary search tree data structure in Section E.4, followed in Section E.5 by how to use it to implement our $\tilde{O}(n\sqrt{r})$ matroid intersection algorithm in the new dynamic-rank-oracle model (the dynamic-*independence*-oracle algorithm is in Section E.12). In Section E.6 we describe our data structure to maintain a basis of a matroid dynamically, and then we use this in our $\tilde{O}_k(n + r\sqrt{r})$ matroid union algorithm in Section E.7 (the special case of k-fold matroid union is in Section E.10). We show our super-linear lower bound in Section E.8. We end our paper with a discussion of open problems in Section E.9.

[15] For example, maintaining a spanning forest in a dynamically changing graph.

In Section E.11 we mention how to implement different matroids oracles in the dynamic-oracle model, and discuss some problems we can solve with our algorithms.

E.2 Technical Overview of Algorithms

E.2.1 The Blocking-Flow Framework

In this section, we give a high-level overview of our algorithms. We will focus on the dynamic-rank-oracle model (Definition E.1.2), and sketch how to efficiently implement the "blocking flow"[16] matroid intersection algorithms of [GS85; Cun86; CLSSW19; Ngu19] in this model. As such, we briefly recap how the $\tilde{O}(n\sqrt{r})$ rank-query (in the traditional oracle model) algorithm of [CLSSW19] works first, and then explain how to implement their framework in the new dynamic oracle model with the same cost.

Their algorithm, like most of the matroid intersection algorithms, is based on repeatedly finding *augmenting paths* in *exchange graphs* (see Section E.3 for a definition). Say we have already found some common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ (we start with $S = \emptyset$). Then the *exchange graph* $G(S)$ is a directed bipartite graph in which finding an (s,t)-path exactly corresponds to increasing the size of S by one. According to Cunningham's blocking-flow argument [Cun86], if we always augment along the *shortest* augmenting path, the lengths of such augmenting paths are non-decreasing. Moreover, if the length of the shortest augmenting path in $G(S)$ is at least $1/\epsilon$, then the size of the current common independent set S must be at least $(1 - O(\epsilon))\, r$ (i.e. it is only $O(\epsilon r)$ away from optimal). Thus, the "blocking flow"-style algorithms consists of two stages:

1. In the first stage, they obtain a $(1-\epsilon)$-approximate solution by finding augmenting paths until their lengths become more than $O(1/\epsilon)$. This is done by running in phases, where in phase i they eliminate all augmenting paths of length $2i$ by finding a so-called "blocking flow"—a maximal (not necessarily maximum) collection of compatible augmenting paths. Each such phase can be implemented using only $\tilde{O}(n)$ rank queries, as shown in [CLSSW19]. This means that the first stage needs a total of $\tilde{O}(n/\epsilon)$ rank queries (in the classic non-dynamic model).

2. In the second stage, they find the remaining $O(\epsilon r)$ augmenting paths one at a time. Each such augmentation can be found in $\tilde{O}(n)$ rank queries, for a total of $\tilde{O}(\epsilon n r)$ queries for this stage.

Using $\epsilon = 1/\sqrt{r}$, [CLSSW19] obtains their $\tilde{O}(n\sqrt{r})$ rank-query exact matroid intersection algorithm. The crux of how to implement the stages efficiently is a binary search trick to explore useful edges of the exchange graph quickly (for e.g. to

[16] Similar to the Hopcroft-Karp's [HK73] bipartite matching and Dinic's [Din70] maximum flow algorithms.

implement a breadth-first-search on the graph). The exchange graph can have up to $\Theta(nr)$ edges in total, but it is not necessary to find all of them. We will argue that this binary search trick (which issues queries far away from each other) can still be implemented in the *dynamic-oracle model*, with the use of some data structures.

E.2.2 Matroid Intersection

Binary Search Tree. The crux of why a breadth-first-search (BFS) and augmenting path searching can be implemented efficiently (in terms of the number of traditional queries) in [CLSSW19] is that they show how to, for $S \in \mathcal{I}$, $u \in S$, and $X \subseteq \bar{S}$, discover an element $x \in X$ with $(S \setminus \{u\}) \cup \{x\} \in \mathcal{I}$ in $O(\log n)$ rank queries using binary search (such a pair (u, x) is called an *exchange pair*, and corresponds to an edge in the exchange graph). The idea is that such an x exists in X if and only if $\text{rank}((S \setminus \{u\}) \cup X) \geq |S|$. Thus, we can do a binary search over X: we split X into two equally-sized subsets X_1 and X_2, and check if such an x exists in X_1 via the above equation. If it does, then we recurse on X_1 to find x. Otherwise, such an x must exist in X_2 (as it does in X), and so we recurse on X_2. To make this process efficient in our new model, we pre-build a binary search tree over the elements of X, where the internal nodes contain all the query-sets we need. That is, in the root node we have the query-set for $S \cup X$, and in its two children for $S \cup X_1$ respectively $S \cup X_2$.

Using this binary tree, one can simulate the binary search process as described above. Since what we need to do in a BFS is to (i) find a replacement element x and (ii) mark x as visited (thus effectively "deactivate" x in X), each time we see x, we just need to remove x from the $O(\log n)$ nodes on a root-to-leaf path, and thus the whole BFS algorithm runs in near-linear time as well.

Batching, Periodic Rebuilding, and Augmenting Sets. The above binary search tree is efficient when the common independent set S is static. However, once we find an augmenting path, we need to update S. This means that every node in the binary search tree needs to be updated. If done naively, this would need at least $\Omega(nr)$ time, as there are up to r augmentations, and rebuilding the tree takes $O(n)$ time. Therefore, we employ a batching approach here. That is, we do not walk through every node and update them immediately when we see an update to S. Instead, we batch k updates (for k to be decided later) and pay an additional $O(k)$-factor every time we want to do a query in our tree. In other words, at some point, we might want to search for exchanges for a common independent set S' (by doing queries like $(S' \setminus \{u\}) \cup X$ to find edges incident to u). Our binary tree might only have an outdated version S (i.e. store sets like $S \cup X$). Then the cost of converting $S \cup X$ to $(S' \setminus \{u\}) \cup X$ is $|S \oplus S'| + 1$, which we assert is less than k. When this number exceeds k, we rebuild the binary search tree completely using the up-to-date S' instead, in $\tilde{O}(n)$ time.

Over the whole run of the algorithm, there are only $O(r \log r)$ updates to our common independent set S (see, e.g., [Cun86; Ngu19]). Hence, the total running

time becomes

$$\underbrace{\tilde{O}(nk\epsilon^{-1})}_{\text{Blocking-flow iterations}} + \underbrace{\tilde{O}(\epsilon rn)}_{\text{Remaining augmenting paths}} + \underbrace{\tilde{O}(nrk^{-1})}_{\text{Rebuilding binary search trees}},$$

which is $\tilde{O}(nr^{2/3})$ for $k = r^{1/3}$ and $\epsilon = r^{-1/3}$.

To achieve the $\tilde{O}(n\sqrt{r})$ bound in our dynamic-rank-oracle model, there is one additional observation we need. By the "Augmenting Sets" argument [CLSSW19], for each element v that we want to query our tree, it suffices to consider changes to S that are in the same distance layer as v is (in a single blocking-flow phase). Since changes to S are uniformly distributed among layers, when the (s,t)-distance in $G(S)$ is d, we only need to spend an additional $O(\frac{k}{d})$-factor (instead of an $O(k)$-factor) when querying the binary search tree. This brings our complexity down to

$$\tilde{O}\left(n\epsilon^{-1} + \sum_{d=1}^{1/\epsilon} \frac{nk}{d}\right) + \tilde{O}(\epsilon rn) + \tilde{O}(nrk^{-1}),$$

where the first part is a harmonic sum which makes for $\tilde{O}(n\epsilon^{-1} + nk)$, and the total running time is $\tilde{O}(n\sqrt{r})$ for $k = r^{1/2}$ and $\epsilon = r^{-1/2}$.

E.2.3 Matroid Union

For simplicity of the presentation in this overview, let's assume we are solving the k-fold matroid union problem and that k—the number of bases[17] we want to find—is constant. A standard black-box reduction from matroid intersection, combined with our algorithm outlined above, immediately gives us an $\tilde{O}(n\sqrt{r})$ bound in the dynamic-rank-oracle model. Nevertheless, we show how to exploit certain properties of matroid union (specifically, the structure of the exchange graphs [EDVJ68; Cun86] resulted from the reduction below) to speed this up to $\tilde{O}(n + r\sqrt{r})$, i.e. *near-linear time* for sufficiently "dense"[18] matroids.

Suppose $\mathcal{M} = (U, \mathcal{I})$ is the matroid we want to find k disjoint bases for. The standard reduction to matroid intersection is that we create k copies of all elements $u \in U$. Then we define two matroids as follows:

- The first matroid $\mathcal{M}_1$ says that we only want to use one version of each element. We set $\mathcal{M}_1 = (U \times \{1, \dots, k\}, \mathcal{I}_{\text{part}})$ to be the partition matroid defined as $S \in \mathcal{I}_{\text{part}}$ if and only if $|\{(u,i) \in S : i = x\}| \leq 1$ for all x.
- The second matroid $\mathcal{M}_2$ says that for each copy of the ground set U we must pick an independent set according to $\mathcal{M}$. That is set $\mathcal{M}_2 = (U \times \{1, \dots, k\}, \hat{\mathcal{I}})$ to be the *disjoint* k-fold union of $\mathcal{M}$, i.e. $S \in \hat{\mathcal{I}}$ if and only if $\{u : (u,i) \in S\}$ is independent in $\mathcal{M}$ for all i.

[17] As an example, consider the problem of finding k disjoint spanning trees of a graph.

[18] We call matroids with $n \gg r$ "dense" by analogy to the graphic matroids where n denotes the number of edges and r the number of vertices.

For a set S which can be partitioned into k disjoint independent sets, notice that in the exchange graph, the number of elements *not* in the first layer is bounded by $O(r)$. This is because every $u \in U$ who is not represented in S will be in the first layer L_1 of the BFS tree. As such, we can build the BFS layers starting from the second layer if we can identify all the elements in this second layer. This can be done by checking for each y *not* in the first layer whether L_1 contains an exchange element of y (via computing the rank of $(S \setminus \{y\}) \cup L_1$; no need to do a binary search). Although binary search is not needed when identifying elements in the second layer, when going backward among layers to find an augmenting path P, we still have to find the exact element in the first layer which can be the first element of P since it will decide which augmenting paths remain "compatible" later. This inspires us to maintain two separate binary search trees: one, of size $O(r)$, for finding edges from the second layer and onward, and the other, of size $O(n)$, for finding the first elements of the augmenting paths. Still, doing a binary search for each element in the first layer results in a total number of $O(r\epsilon^{-1})$ queries to the binary search tree, which is too much. To reduce the number of queries down to $\tilde{O}(r)$, we note that only binary searches which correspond to the actual augmenting paths will succeed, i.e., reach the leaf nodes of the binary search tree. Since there are at most $O(r/d)$ augmenting paths when the (s,t)-distance in $G(S)$ is d, we only need to do $O(r/d)$ queries to the binary search tree; other queries can be blocked by first checking if their corresponding exchange elements exist in the first layer. This results in a running time of $\tilde{O}(n + r\sqrt{n})$ (note: $\sqrt{n}$ and not $\sqrt{r}$), which already matches Gabow's algorithm for k-disjoint spanning tree [GW88].

Toward $\tilde{O}(n + r\sqrt{r})$ for Matroid Union. The bottleneck of the above algorithm is that we need to do binary searches over (and hence rebuild periodically) the tree data structure for the first layer (of size $\Omega(n)$). If we can reduce the size of this tree down to $O(r)$, then the running time would be $\tilde{O}(n + r\sqrt{r})$. This suggests that we might want to somehow "sparsify" the first layer. Indeed, for a single augmenting path, we only need a basis of the first layer. As a concrete example, consider the case of a graphic matroid: Given a forest S, an edge $e \in S$, and the set of non-tree edges $E \setminus S$, we want to find a "replacement" edge e' in $E \setminus S$ for e which "restores" the connectivity of $S - e + e'$. In this case, it suffices to only consider a spanning forest (i.e. "basis") B of $E \setminus S$, in the sense that such a replacement edge exists in $E \setminus S$ if and only if it exists in this spanning forest $B \subseteq E \setminus S$.

Moreover, note that after each augmentation a single element will be removed from the first layer. Thus, if we can maintain a decremental basis of the first layer, we can build our binary search tree data structure dynamically on top of this basis and get the desired time bound.

Maintaining a Basis in a Matroid. Our data structure for maintaining a basis is inspired by the dynamic minimum spanning tree algorithm of [Fre85], in combination with the sparsification trick of [EGIN97]. It uses $\tilde{O}(n)$ time to initialize,

and then $\tilde{O}(\sqrt{r})$ dynamic rank queries[19] per deletion. It also supports maintaining a min-weight basis.

Let $L \subseteq U$ for $|L| = O(n)$ be the first layer in which we want to maintain a dynamic basis. In the preprocessing stage, we split L into $\sqrt{n}$ blocks $L_1, L_2, \ldots, L_{\sqrt{n}}$ of size roughly $\sqrt{n}$ and compute the basis of L from left to right. We also build the "prefix sums" of these $\sqrt{n}$ blocks so that we can quickly access/query sets of the form $L_1 \cup L_2 \cup \cdots \cup L_k$ for all values of k. When we remove an element x from L_i, we first update the prefix sums in $O(\sqrt{n})$ time. If x is not in the basis we currently maintain, then nothing additional needs to be done. Otherwise, we have to find the "first" replacement element, which is guaranteed to be located in blocks $L_i, \ldots, L_{\sqrt{n}}$. The block L_j in which the replacement element lies can be identified simply by inspecting the ranks of the prefix sums, and after that, we then go through elements in that block to find the exact element. Note that blocks after L_j need not be updated, as for them it does not matter what basis we picked among blocks L_1 to L_j. This gives us an $O(\sqrt{n})$-update-time algorithm for maintaining a basis of a matroid.

To get a complexity of $O(\sqrt{r}\log n)$, we show that a similar sparsification structure as that of [EGIN97] for dynamic graph algorithms also works for arbitrary matroids. The sparsification is a balanced binary tree over the n elements, where in each node we have an instance of our (un-sparsified) underlying data structure to maintain a basis consisting of elements in the subtree rooted at the node. Only elements part of the basis of a node are propagated upwards to the parent node. This means that in each instance of our underlying data structure we work over a ground set of size at most $2r$. Thus, each update corresponds to at most two updates (a single insertion and deletion) to at most $O(\log n)$ (which is the height of the tree) nodes of the tree, each costing $O(\sqrt{r})$ dynamic rank queries in order to maintain the basis at this node. This results in the desired time bound.

E.3 Preliminaries

Notation. We use standard set notation. In addition to that, for two sets X and Y, we use $X + Y$ to denote $X \cup Y$ (when $X \cap Y = \emptyset$) and $X - Y$ to denote $X \setminus Y$ (when $Y \subseteq X$). For an element v, $X + v$ and $X - v$ refer to $X + \{v\}$ and $X - \{v\}$, respectively. Let $X \oplus Y$ denote the symmetric difference of X and Y.

Matroid. In this paper, we use the standard notion of matroids which is defined as follows.

Definition E.3.1. A *matroid* $\mathcal{M} = (U, \mathcal{I})$ is defined by a tuple consisting of a finite *ground set* U and a non-empty family of *independent sets* $\mathcal{I} \subseteq 2^U$ such that the following properties hold.

[19] In the application of our matroid union algorithm, there will only be $\tilde{O}(r)$ updates, so this is efficient enough for our final $\tilde{O}(n + r\sqrt{r})$ algorithm.

- **Downward closure:** If $S \in \mathcal{I}$, then any subset $S' \subseteq S$ is also in $\mathcal{I}$.
- **Exchange property:** For any two sets $S_1, S_2 \in \mathcal{I}$ with $|S_1| < |S_2|$, there exists an $x \in S_2 \setminus S_1$ such that $S_1 + x \in \mathcal{I}$.

Let U be the ground set of a matroid $\mathcal{M}$. For $S \subseteq U$, let $\bar{S}$ denote $U \setminus S$. For $X \subseteq U$, the *rank* of X, denoted by $\text{rank}(X)$, is the size of the largest independent set contained in X, i.e., $\text{rank}(X) = \max_{S \in \mathcal{I}} |X \cap S|$. The *rank* of a matroid $\mathcal{M} = (U, \mathcal{I})$ is the rank of U. We let $r \leq n$ denote the rank of the input matroids. When the input consists of more than one matroid (e.g., in the matroid union problem), let rank_i denote the rank function of the i^{th} matroid. A *basis* of X is an independent set $S \subseteq X$ with $|S| = \text{rank}(X)$. A basis of $\mathcal{M}$ is a basis of U. The *span* of X contains elements whose addition to X does not increase the rank of it, i.e., $\text{span}(X) = \{u \in U \mid \text{rank}(X \cup \{u\}) = \text{rank}(X)\}$.

Fact E.3.2. *The rank function is submodular. That is,* $\text{rank}(X) + \text{rank}(Y) \geq \text{rank}(X \cap Y) + \text{rank}(X \cup Y)$ *holds for each* $X, Y \subseteq U$.

Fact E.3.3 (see, e.g., [Pri15, Lemma 1.3.6])**.** $\text{rank}(A) = \text{rank}(\text{span}(A))$ *holds for every* $A \subseteq U$.

Lemma E.3.4. *For two sets* X, Y *and their bases* S_X, S_Y*, it holds that* $\text{rank}(S_X + S_Y) = \text{rank}(X + Y)$.

Proof. Since $X, Y \subseteq \text{span}(S_X + S_Y)$, we have $S_X + S_Y \subseteq X + Y \subseteq \text{span}(S_X + S_Y)$. The lemma then follows from $\text{rank}(S_X + S_Y) = \text{rank}(\text{span}(S_X + S_Y))$ using Fact E.3.3. □

Exchange Graph. Our algorithms for matroid intersection and union will be heavily based on finding augmenting paths in exchange graphs.

Definition E.3.5 (Exchange Graph)**.** For two matroids $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$ over the same ground set and an $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* with respect to S is a directed bipartite graph $G(S) = (U \cup \{s, t\}, E_S)$ with $s, t \notin U$ being two distinguished vertices and $E_S = E_1 \cup E_2 \cup E_s \cup E_t$, where

$$
\begin{aligned}
E_1 &= \{(x, y) \mid x \in S, y \notin S, \text{ and } S - x + y \in \mathcal{I}_1\},\\
E_2 &= \{(y, x) \mid x \in S, y \notin S, \text{ and } S - x + y \in \mathcal{I}_2\},\\
E_s &= \{(s, x) \mid S + x \in \mathcal{I}_1\}, \text{ and}\\
E_t &= \{(x, t) \mid S + x \in \mathcal{I}_2\}.
\end{aligned}
$$

The *distance layers* of $G(S)$ is the sets $L_1, \ldots, L_{d_{G(S)}(s,t)-1}$, where L_ℓ consists of elements in U that are of distance ℓ from s in $G(S)$. Most matroid intersection algorithms including ours are based on augmenting a common independent set with an augmenting path in $G(S)$ until such a path does not exist. The following lemma certifies the correctness of this approach.

Lemma E.3.6 (Augmenting Path)**.** *Let P be a shortest (s,t)-path[20] of $G(S)$. Then, the set $S' := S \oplus (V(P) \setminus \{s,t\})$ is a common independent set with $|S'| = |S| + 1$. On the other hand, if t is unreachable from s in $G(S)$, then S is a largest common independent set.*

We write $S \oplus P$, where P is an augmenting path in $G(S)$, for the common independent set $S' := S \oplus (V(P) \setminus \{s,t\})$ obtained by augmenting S along P. Let $d_{G(S)}(u,v)$ denote the (u,v)-distance in $G(S)$. When S is clear from context, let d_t denote $d_{G(S)}(s,t)$. The following lemma states that if $d_{G(S)}(s,t)$ is large, then S is close to being optimal.

Lemma E.3.7 ([Cun86])**.** *If $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ satisfies $d_{G(S)}(s,t) \geq d$, then $|S|$ is at least $\left(1 - O(\frac{1}{d})\right) r$.*

The following bound on the total length of shortest augmenting paths will be useful for our analysis.

Lemma E.3.8 ([Cun86])**.** *If we solve matroid intersection by repeatedly finding the shortest augmenting paths, then the sum of the lengths of these augmenting paths is $O(r \log r)$.*

Lemma E.3.9 ([Cun86; Pri15; CLSSW19])**.** *If we augment along a shortest (s,t)-path in $G(S)$ to obtain S', then for each $u \in U$, the following hold (let $d := d_{G(S)}$ and $d' := d_{G(S')}$).*

1. *If $d(s,u) < d(s,t)$, then $d'(s,u) \geq d(s,u)$. If $d(u,t) < d(s,t)$, then $d'(u,t) \geq d(u,t)$.*
2. *If $d(s,u) \geq d(s,t)$, then $d'(s,u) \geq d'(s,t)$. If $d(u,t) \geq d(s,t)$, then $d'(u,t) \geq d'(s,t)$.*

Augmenting Sets. The following notion of *augmenting sets*, introduced by [CLSSW19], models a collection of "mutually compatible" augmenting paths, i.e., paths that can be augmented sequentially without interfering with each other.

Definition E.3.10 (Augmenting Set [CLSSW19, Definition 24])**.** Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ satisfy $d_{G(S)}(s,t) = d_t$ and let $L_1, L_2, \ldots, L_{d_t-1}$ be the distance layers of $G(S)$. A collection $\Pi := (D_1, D_2, \cdots, D_{d_t-1})$ is an *augmenting set* in $G(S)$ if

[20] In fact, P only needs to be "chordless" [BBMN21], i.e., without shortcuts. Nonetheless, a shortest (s,t)-path suffices for our rank-query algorithms.

(i) $D_\ell \subseteq L_\ell$ holds for each $1 \leq \ell < d_t$,
(ii) $|D_1| = |D_2| = \cdots = |D_{d_t-1}|$,
(iii) $S + D_1 \in \mathcal{I}_1$,
(iv) $S + D_{d_t-1} \in \mathcal{I}_2$,
(v) $S - D_\ell + D_{\ell+1} \in \mathcal{I}_1$ holds for each *even* $1 \leq \ell < d_t - 1$, and
(vi) $S - D_{\ell+1} + D_\ell \in \mathcal{I}_2$ holds for each *odd* $1 \leq \ell < d_t - 1$.

One can think of the concept of augmenting sets as a generalization of augmenting paths. Indeed, an augmenting path is an augmenting set where $|D_1| = \cdots = |D_{d_t-1}| = 1$. The term "mutually compatible" augmenting paths is formalized as follows.

Definition E.3.11 (Consecutive Shortest Paths [CLSSW19, Definition 28])**.** A collection of vertex-disjoint shortest (s,t)-paths $\mathcal{P} = (P_1, \ldots, P_k)$ in $G(S)$ is a collection of *consecutive shortest paths* if P_i is a shortest augmenting path in $G(S \oplus P_1 \oplus \cdots \oplus P_{i-1})$ for each $1 \leq i \leq k$.

The following structural lemmas of [CLSSW19] will be useful for us, particularly in deriving Lemma E.5.4 in Section E.5.

Lemma E.3.12 ([CLSSW19, Theorem 25])**.** *Let $\Pi := (D_1, \ldots, D_{d_t-1})$ be an augmenting set in $G(S)$. Then, $S' := S \oplus \Pi := S \oplus D_1 \oplus \cdots \oplus D_{d_t-1}$ is a common independent set.*

For two augmenting sets $\Pi = (D_1, D_2, \ldots, D_{d_t-1})$ and $\Pi' = (D'_1, D'_2, \ldots, D'_{d_t-1})$, we use $\Pi \subseteq \Pi'$ to denote that $D_\ell \subseteq D'_\ell$ hold for each $1 \leq \ell < d_t$. In this case, let $\Pi' \setminus \Pi := (D'_1 \setminus D_1, \ldots, D'_{d_t-1} \setminus D_{d_t-1})$. We will hereafter abuse notation and let Π also denote the set of elements $D_1 \cup \cdots \cup D_{d_t-1}$ in it. In particular, $U \setminus \Pi$ denotes $U \setminus (D_1 \cup \cdots \cup D_{d_t-1})$.

Lemma E.3.13 ([CLSSW19, Theorem 33])**.** *For two augmenting sets $\Pi \subseteq \Pi'$ in $G(S)$, $\Pi' \setminus \Pi$ is an augmenting set in $G(S \oplus \Pi)$.*

Lemma E.3.14 ([CLSSW19, Theorem 29])**.** *Given a collection of consecutive shortest paths $P_1, \ldots, P_k$ in $G(S)$, where $P_i = (s, a_{i,1}, \ldots, a_{i,d_t-1}, t)$, the collection $\Pi = (D_1, \ldots, D_{d_t-1})$, where $D_i = \{a_{1,i}, \ldots, a_{k,i}\}$, is an augmenting set in $G(S)$.*

The converse of Lemma E.3.14 also holds.

Lemma E.3.15 ([CLSSW19, Theorem 34])**.** *Given an augmenting set Π in $G(S)$, there is a collection consecutive shortest paths $P_1, \ldots, P_k$ in $G(S)$ where $P_i = (s, a_{i,1}, \ldots, a_{i,d_t-1}, t)$ such that $D_i = \{a_{1,i}, \ldots, a_{k,i}\}$.*

Remark E.3.16. Note that Lemmas E.3.14 and E.3.15 are not equivalent to the exact statements of [CLSSW19, Theorems 29 and 34] (in particular, they did not specify how Π and P_i are constructed), but our versions are clear from their proof.

Claim E.3.17. *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $d_{G(S)}(s,t) = d_t$ and $\Pi \subseteq \Pi'$ be two augmenting sets in $G(S)$. Let $u \notin \Pi'$ be an element. If u is not on any augmenting path of length d_t in $G(S \oplus \Pi)$, then u is not on any augmenting path of length d_t in $G(S \oplus \Pi')$ either.*

Proof. Let $S' := S \oplus \Pi$ and $S'' := S \oplus \Pi'$. Since S'' can be obtained by augmenting S' along a series of shortest augmenting paths (by Lemmas E.3.13 and E.3.15), the claim follows from the fact that the (s,u)-distance and (u,t)-distance are monotonic (Lemma E.3.9). □

Claim E.3.18. *Let $\Pi = (D_1, \ldots, D_{d_t-1})$ be an augmenting set in $G(S)$ and $P = (s, a_1, \ldots, a_{d_t-1}, t)$ be an augmenting path in $G(S \oplus \Pi)$. Then, $\Pi' = (D_1 + a_1, \ldots, D_{d_t-1} + a_{d_t-1})$ is an augmenting set in $G(S)$.*

Proof. This directly follows from Lemmas E.3.14 and E.3.15. □

Using Dynamic Oracle. In the following sections except for Section E.12 where we discuss independence-query algorithms, all algorithms and data structures will run in the *dynamic-rank-oracle* model (see Definition E.1.2). In other words, we will simply write "in t time" for "in t time and dynamic rank queries". We will use the term *query-sets* to refer to the sets S_i in Definition E.1.2. In particular, constructing a query-set means building the corresponding set from $S_0 = \emptyset$ with the INSERT$(\cdot)$ operation. Insertion/Deletion of an element into/from a query-set is done via the INSERT/DELETE operations. Using the QUERY operation, we assume that we know the ranks of all the query-sets we construct in our algorithms.

E.4 Binary Search Tree

In this section, we give the core data structure of our algorithms which allows us to do binary searches and find free elements (elements x such that $S + x \in \mathcal{I}$) and exchange pairs (pairs (x,y) such that $S - x + y \in \mathcal{I}$, corresponding to edges in the exchange graph) efficiently. We also support updating the common independent set S that the exchange relationship is based upon. For a matroid $\mathcal{M} = (U, \mathcal{I})$, the data structure has the following guarantee ($s, t \notin U$ denote the two distinguished vertices of the exchange graph as defined in Definition E.3.5).

Theorem E.4.1. *For any integer $\beta \geq 1$, there exists a data structure that supports the following operations.*

- INITIALIZE$(\mathcal{M}, S, Q_S, X)$*: Given $S \in \mathcal{I}$, the query-set Q_S that corresponds to S, and $X \subseteq \bar{S}$ (respectively, $X \subseteq S$ or $X = \{t\}$), initialize the data structure in $\tilde{O}(|X|)$ time. The data structure also maintains S.*
- FIND(y)*: Given $y \in S \cup \{s\}$ (respectively, $y \in \bar{S}$),*

  - *if $y \in S$ (respectively, $X \subseteq S$), then return an $x \in X$ such that $S-y+x \in \mathcal{I}$ (respectively, $S - x + y \in \mathcal{I}$), or*
  - *if $y = s$ (respectively, $X = \{t\}$), then return an $x \in X$ such that $S+x \in \mathcal{I}$ (respectively, return the only element $x = s$ or $x = t$ in X if $S + y \in \mathcal{I}$ and $\perp$ otherwise).*

  The procedure returns $\perp$ if such an x does not exist. The procedure takes $\tilde{O}(\beta)$ time if the result is not $\perp$, and $\tilde{O}(1)$ time otherwise.

- FIND(x)*: Given $x \in X$, if $x \notin \{s, t\}$, delete x from X in $O(\log n)$ time.*
- REPLACE(x, y)*: Given $x \in X$ and $y \notin X$, replace x in X by y in $O(\log n)$ time.*
- UPDATE(Δ)*: Update S to $S \oplus (\Delta \setminus \{s, t\})$ in amortized $\tilde{O}(\frac{|X|\cdot|\Delta|}{\beta})$ time.*

Remark E.4.2. To make sense of the seemingly complicated input and casework of Theorem E.4.1, one should focus on the first item of FIND$(\cdot)$. We will use Theorem E.4.1 to explore the exchange graphs, and thus we need to find an exchange element x of y as in the first case. The additional complication is included solely because we also have to deal with edges incident to s or t. For instance, say $X \subseteq \bar{S}$, then FIND(s) finds an edge in $G(S)$ directed from s to X. This will make our algorithms presented later cleaner (see Algorithm E.1 for example).

Sometimes, we will omit the Q_S parameter of INITIALIZE, meaning that we explicitly build the query-set Q_S from S in $O(r)$ time before running the actual initialization. In such cases, $|X|$ will be $\Omega(r)$, and thus this incurs no overhead.

We will later refer to the case of $X \subseteq \bar{S}$ as the *co-circuit binary search tree* and the case of $X \subseteq S$ as the *circuit binary search tree*. The data structure follows from the binary search algorithm of [CLSSW19, Lemma 10], which is based on the following observation.

Observation E.4.3 ([CLSSW19])**.** *To find free elements and exchange pairs, we can use the following observations.*

(i) Free element*: There exists an $x \in X$ such that $S + x \in \mathcal{I}$ if and only if* $\operatorname{rank}(S + X) > |S|$.

(ii) Co-circuit exchange*: Given $y \in S$, there exists an $x \in X$ such that $S-y+x \in \mathcal{I}$ if and only if* $\operatorname{rank}(S - y + X) \geq |S|$.

(iii) Circuit exchange*: Given $y \notin S$, there exists an $x \in X$ such that $S - x + y \in \mathcal{I}$ if and only if* $\operatorname{rank}(S - X + y) = |S - X + y|$.

The data structure of Theorem E.4.1 is built upon the following similar data structure whose independent set S is "static" in the sense that its update will be specified for each query. We construct the data structure of Lemma E.4.4 first, and then use it for Theorem E.4.1 later in the section.

Lemma E.4.4. *There exists a data structure that supports the following operations.*

- INITIALIZE$(\mathcal{M}, S, Q_S, X)$*: Given* $S \in \mathcal{I}$*, a query-set* Q_S *corresponding to* S*, and* $X \subseteq \bar{S}$ *(respectively,* $X \subseteq S$ *or* $X = \{t\}$*), initialize the data structure in* $\tilde{O}(|X|)$ *time.*
- FIND(y, Δ)*: Given* $\Delta \subseteq V$*, let* $S' := S \oplus \Delta$*. It is guaranteed that* $S' \in \mathcal{I}$*. Given* $y \in S' \cup \{s\}$ *(respectively,* $y \in \bar{S}'$*),*
  - *if* $y \in S'$ *(respectively,* $X \subseteq S'$*), then return an* $x \in X$ *such that* $S' - y + x \in \mathcal{I}$ *(respectively,* $S' - x + y \in \mathcal{I}$*), otherwise*
  - *if* $y = s$ *(respectively,* $X = \{t\}$*), then return an* $x \in X$ *such that* $S' + x \in \mathcal{I}$ *(respectively, return the only element* $x = s$ *or* $x = t$ *in* X *if* $S' + y \in \mathcal{I}$ *and* $\perp$ *otherwise),*

  in $\tilde{O}(\beta)$ *time. The procedure returns* $\perp$ *if such an* x *does not exist.*
- DELETE(x)*: Given* $x \in X$*, delete* x *from* X *in* $O(\log n)$ *time.*
- REPLACE(x, y)*: Given* $x \in X$ *and* $y \notin X$*, replace* x *by* y *in* X *in* $O(\log n)$ *time.*

We present the co-circuit version of the data structures as the circuit version is analogous (their difference is essentially stated in the two cases (ii) and (iii) of Observation E.4.3). The data structure of Lemma E.4.4 is a balanced binary tree in which every node v corresponds to a subset X_v of X. The subsets corresponding to nodes at the same level form a disjoint partition of X. There are $|X|$ leaf nodes, each of which corresponds to a single-element subset of X. An internal node v with children u_1 and u_2 has $X_v = X_{u_1} \sqcup X_{u_2}$. Each node v is also associated with a query-set $Q_v := S + X_v$, for which we have prepared a dynamic oracle (see Definition E.1.2).

Initialization. In the initialization stage, we first compute the query-set of the root node $Q_r := S + X$ from Q_S in $O(|X|)$ time. As long as the current node v has $|X_v| > 1$, we split X_v into two equally-sized subsets X_{u_1}, X_{u_2}, compute Q_{u_1}, Q_{u_2} from Q_v, and then recurse on the two newly created nodes u_1 and u_2. Computing Q_{u_1} and Q_{u_2} from Q_v takes $O(|X_v|)$ time in total, and thus the overall running time for initialization is $\tilde{O}(|X|)$.

Query. To find an exchange element of $y \in S'$, we perform a binary search on the tree. For each node v, we can test whether such an element exists in X_v via Observation E.4.3(ii) by computing the query-set $Q'_v := S' - y + X_v$ from $Q_v := S + X_v$ in $1 + |\Delta|$ dynamic-oracle queries. If such an element does not exist for the root $X_r = X$, then we return $\perp$. Otherwise, for node v initially being r, there must exist one of the child nodes u_i of v where such an exchange x exists in

X_{u_i}. We then recurse on u_i until we reach a leaf node, at which point we simply return the corresponding element. Similarly, to find a free element, we compute the rank of $Q'_v := S' + X_v$ instead (see Observation E.4.3(i)). Since we need to compute Q'_v for each of the visited nodes, the running time is $O(|\Delta| \log n)$.

Update. For deletion of x, we simply walk up from the leaf node corresponding to x to the root node and remove x from each of the X_v and Q_v. This takes time proportional to the depth of the tree, which is $O(\log n)$. Replacement of x by y follows similarly from deletion of x: instead of simply removing x from X_v and Q_v, we add y to them as well.

Remark E.4.5. Note that the above binary search tree is static in the sense that we only deactivate elements from a fixed initial set. We can extend this data structure to support a dynamically changing input set X by using a dynamic binary search tree based on partial rebuilding [And89; And91] instead. The amortized time complexity remains the same since rebuilding a subtree takes time proportional to the number of nodes of it.

E.4.1 Periodic Rebuilding

Here we extend Lemma E.4.4 to prove Theorem E.4.1. Recall that the difference between the two data structures is that we need to support a dynamically changing independent set in Theorem E.4.1 (which we will need since S changes after each augmentation in our matroid algorithms). How we achieve this is to essentially employ a batch-and-rebuild approach to the binary search tree of Lemma E.4.4.

Proof. We maintain a binary search tree $\mathcal{T}$ constructed as $\textsc{Initialize}(\mathcal{M}, S, Q_S, X)$ of Lemma E.4.4 and a collection of "batched" updates Δ_{batch} of size at most β. Throughout the updates, we also maintain the query-set corresponding to the current S starting from the given Q_S and the query-set corresponding to $S + X$, which initially can be computed from Q_S in $O(|X|)$ time. Each call to $\textsc{Find}(y)$ is delegated to $\mathcal{T}.\textsc{Find}(y, \Delta_{\text{batch}})$, which runs in time $\tilde{O}(|\Delta_{\text{batch}}|) = \tilde{O}(\beta)$ time. Note that we can test whether the result of $\mathcal{T}.\textsc{Find}(\cdot)$ will be $\perp$ in $\tilde{O}(1)$ time by simply checking if Observation E.4.3(ii) (or (i) if $y = s$) holds with the query-set corresponding to $S + X$ we maintain.

Each call to $\textsc{Delete}(x)$ and $\textsc{Replace}(x, y)$ translates simply to $\mathcal{T}.\textsc{Delete}(x)$ and $\mathcal{T}.\textsc{Replace}(x, y)$. For an update to S with Δ, we set $\Delta_{\text{batch}} \leftarrow \Delta_{\text{batch}} \cup \Delta$ and update S and the query-sets accordingly. If the size of Δ_{batch} exceeds β, then we rebuild the binary search tree with the input common independent set being the up-to-date S we maintain. Note that we will pass query-set Q_S to $\textsc{Initialize}$ to not pay the extra $O(r)$ factor. Finally, since the binary search tree is now up-to-date, we set Δ_{batch} to be $\emptyset$. The rebuilding takes $\tilde{O}(|X|)$ time and is amortized to $\tilde{O}(\frac{|X| \cdot |\Delta|}{\beta})$ per update operation with $|\Delta|$ changes. □

E.5 Matroid Intersection

In this section, we present a matroid intersection algorithm in the dynamic-rank-oracle model that matches the state-of-the-art algorithm [CLSSW19] in the traditional model.

Theorem E.5.1. *For two matroids $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$, it takes $\tilde{O}(n\sqrt{r})$ time to obtain the largest $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ in the dynamic-rank-oracle model.*

The algorithm follows the blocking-flow framework of [CLSSW19] similar to the Hopcroft-Karp algorithm for bipartite matching [HK73], which goes as follows. Initially, they start with $S = \emptyset$.

1. First, they obtain a common independent set that is of size at least $(1 - \epsilon)r$ by eliminating all augmenting paths of length $O(1/\epsilon)$. In each of the $O(1/\epsilon)$ iterations, they first compute the distance layers of $G(S)$ along which they find a maximal set of compatible shortest augmenting paths using an approach similar to a depth-first-search from s. Augmenting paths are searched in a depth-first-search manner. Whenever an element has no out-edge with respect to the current common independent set to the next layer, they argue that it can be safely removed as it will not be on a shortest augmenting path anymore in this iteration. Augmenting along these augmenting paths increases the (s, t)-distance of $G(S)$ by at least one.

2. With the current solution which is only ϵ fraction away from being optimal, they find the remaining $O(\epsilon r)$ augmenting paths one at a time.

A proper choice of ϵ (in this case it is $\epsilon = 1/\sqrt{r}$) that balances the cost between the two steps results in their algorithm.

E.5.1 Building Distance Layers

Building distance layers and finding a single augmenting path in Step 2 is immediate by replacing binary searches in [CLSSW19, Algorithm 4] with the binary search trees of Theorem E.4.1.

Lemma E.5.2. *It takes $\tilde{O}(n)$ time to compute the (s, u)-distance for each $u \in U$ and find the shortest (s, t)-path in $G(S)$ or determine that t is unreachable from s.*

Proof. First, we build two binary search trees via Theorem E.4.1 with $\beta = 1$, a circuit binary search tree $\mathcal{T}_1 := \textsc{Initialize}(\mathcal{M}_1, S, X_1)$ where $X_1 = S$ for the first matroid and a co-circuit binary search tree $\mathcal{T}_2 := \textsc{Initialize}(\mathcal{M}_2, S, X_2)$ where $X_2 = \bar{S}$ for the second matroid. Initializing these takes $\tilde{O}(n)$ time. These two binary search trees allow us to explore the exchange graph efficiently.

Then we run the usual BFS algorithm from the source s (or equivalently, all $u \in \bar{S}$ with $S + u \in \mathcal{I}_1$). For each visited element u, if $u \in S$, then we repeatedly

find $x \in X_2$ such that $S - u + x \in \mathcal{I}_2$ using $\mathcal{T}_2$.FIND(u), mark x as visited, and remove x from X_2 via $\mathcal{T}_2$.DELETE(x) (until $\perp$ is returned). Similarly, for $u \in \bar{S}$, we find $x \in X_1$ with $S - x + u \in \mathcal{I}_1$, mark x as visited, and remove x from X_1 using $\mathcal{T}_1$. This explores all the unvisited out-neighbors of u in $G(S)$. Since each element will be visited at most once, the total running time is $\tilde{O}(n)$. □

E.5.2 Blocking Flow

In this section, we prove the following lemma regarding a single phase of blocking-flow computation.

Lemma E.5.3. *Given an $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with s,t-distance $d_{G(S)}(s,t) = d$, it takes $\tilde{O}\left(n + \frac{n\sqrt{r}}{d} + \frac{(|S'|-|S|)\cdot nd}{\sqrt{r}}\right)$ time to obtain an $S' \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $d_{G(S')}(s,t) > d_{G(S)}(s,t)$.*

Before proceeding to prove Lemma E.5.3, we first use it to finish our matroid intersection algorithm. Like Hopcroft-Karp bipartite matching algorithm [HK73] and the matroid intersection algorithm of [CLSSW19], we run several iterations of blocking-flow, and then keep augmenting until we get the optimal solution.

Proof of Theorem E.5.1. Starting from an empty set $S = \emptyset$, we run the blocking-flow algorithm until $d_{G(S)}(s,t) \geq \sqrt{r}$. This, by Lemma E.5.3, takes

$$\tilde{O}\left(n\sqrt{r}\right) + \tilde{O}\left(\sum_{d=1}^{\sqrt{r}} \frac{n\sqrt{r}}{d}\right) + \tilde{O}\left(\frac{n}{\sqrt{r}} \cdot \left(\sum_{d=1}^{\sqrt{r}} d \cdot (|S_d| - |S_{d-1}|)\right)\right) \tag{E.1}$$

time, where S_d is the size of the S we get after augmenting along paths of length d. Observe that $\sum_{d=1}^{\sqrt{r}} d \cdot (|S_d| - |S_{d-1}|)$ is the sum of lengths of the augmenting paths that we use, and thus the third term in Equation (E.1) is $\tilde{O}(\frac{n}{\sqrt{r}} \cdot r) = \tilde{O}(n\sqrt{r})$ by Lemma E.3.8. The second term also sums up to $\tilde{O}(n\sqrt{r})$ (by a harmonic sum), and therefore the total running time of the blocking-flow phases is $\tilde{O}(n\sqrt{r})$. The current common independent set S has size at least $r - O(\sqrt{r})$ by Lemma E.3.7, and thus finding the remaining $O(\sqrt{r})$ augmenting paths one at a time takes a total running time of $\tilde{O}(n\sqrt{r})$ via Lemma E.5.2. This concludes the proof of Theorem E.5.1. □

The rest of the section is to prove Lemma E.5.3. Our blocking-flow algorithm is a slight modification to [CLSSW19, Algorithm 5], as shown in Algorithm E.1. It takes advantage of the data structure of Theorem E.4.1 to explore an out-edge from the current element a_ℓ to $A_{\ell+1}$—the set of "alive" elements in the next layers—while (approximately) keeping track of the current common independent set S. An element u is "alive" if it has not been included in the augmenting set $\Pi := (D_1, \ldots, D_{d_t-1})$ yet, nor has the algorithm determines that there cannot be any shortest augmenting path through u.

Algorithm E.1: Blocking flow

Input: $S \in \mathcal{I}_1 \cap \mathcal{I}_2$
Output: $S' \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $d_{G(S')}(s,t) > d_{G(S)}(s,t)$

Build the distance layers $L_1, \ldots, L_{d_t - 1}$ of $G(S)$ with Lemma E.5.2
$L_0 \leftarrow \{s\}$ and $L_{d_t} \leftarrow \{t\}$
$A_\ell \leftarrow L_\ell$ for each $0 \leq \ell \leq d_t$
$\mathcal{T}_\ell \leftarrow \textsc{Initialize}(\mathcal{M}_1, S, Q_S, L_\ell)$ for each *odd* $1 \leq \ell \leq d_t$ by Theorem E.4.1 with $\beta = \sqrt{r}/d$
$\mathcal{T}_\ell \leftarrow \textsc{Initialize}(\mathcal{M}_2, S, Q_S, L_\ell)$ for each *even* $1 \leq \ell \leq d_t$ by Theorem E.4.1 with $\beta = \sqrt{r}/d$
$\ell \leftarrow 0$, $a_0 \leftarrow s$, and $D_\ell \leftarrow \emptyset$ for each $1 \leq \ell < d_t$
while $\ell \geq 0$ **do**
  if $\ell < d_t$ **then**
    if $A_\ell = \emptyset$ **then break**
    $a_{\ell+1} \leftarrow \mathcal{T}_{\ell+1}.\textsc{Find}(a_\ell)$
    if $a_{\ell+1} = \perp$ **then**
      $\mathcal{T}_\ell.\textsc{Delete}(a_\ell)$
      $A_\ell \leftarrow A_\ell - a_\ell$ and $\ell \leftarrow \ell - 1$
    else
      $\ell \leftarrow \ell + 1$
  else
    // Found augmenting path $a_1, a_2, \ldots a_\ell$
    for $i \in \{1, 2, \ldots, d_t - 1\}$ **do**
      $D_i \leftarrow D_i + a_i$ and $A_i \leftarrow A_i - a_i$
      $\mathcal{T}_i.\textsc{Delete}(a_i)$ and $\mathcal{T}_i.\textsc{Update}(\{a_{i-1}, a_i\})$
    $\ell \leftarrow 0$
return $S' := S \oplus \Pi$, where $\Pi := (D_1, D_2, \ldots, D_{d_t - 1})$

We emphasize that the difference between Algorithm E.1 and [CLSSW19, Algorithm 5] is exactly in the replacement of binary searches with the data structure of Theorem E.4.1. Note that indeed by the specification stated in Theorem E.4.1, the binary search trees let us explore edges in the exchange graph (see Remark E.4.2). As a result, our proof will focus on showing that such a replacement does not affect the correctness. For this, we need the concept of *augmenting sets* (see Definition E.3.10) which characterizes a collection of "mutually compatible" augmenting paths—i.e. a "blocking flow". The structural results in Section E.3 culminate in the following lemma that is key to the correctness of our algorithm. It models when we can safely "remove" an element since there will be no augmenting path through it in the future. This is in particular required for us (as opposed to the simpler argument used in [CLSSW19]) because the set S is not fully updated after each augmentation (at

least in the binary search trees that we use to explore the exchange graphs).

Lemma E.5.4. *Let $\Pi \subseteq \Pi'$ be augmenting sets in $G(S)$ with distance layers $L_1, \ldots, L_{d_t-1}$ where $d_{G(S)}(s,t) = d_t$. For $x \in L_\ell$, if there is no $y \in L_{\ell+1}$ such that*

$$(S \oplus D_\ell \oplus D_{\ell+1}) \oplus \{x, y\} \in \mathcal{I}, \text{ where } \mathcal{I} := \begin{cases} \mathcal{I}_1, & \text{if } \ell \text{ is even} \\ \mathcal{I}_2, & \text{if } \ell \text{ is odd} \end{cases}, \tag{E.2}$$

then there is no augmenting path of length d_t through x in $G(S \oplus \Pi')$.

Proof. We claim that there is no augmenting path of length d_t through x in $G(S \oplus \Pi)$: If there is such a P, then we can put P into Π and get an augmenting set $\tilde{\Pi} := (\tilde{D}_1, \ldots, \tilde{D}_{d_t-1})$ by Claim E.3.18. By definition of the augmenting set, this means that there is such a $y \in \tilde{D}_{k+1} \setminus D_{k+1}$ satisfying (E.2), a contradiction to our assumption. The lemma now follows from Claim E.3.17. □

We are now ready to prove Lemma E.5.3.

Proof of Lemma E.5.3. First, We analyze the running time of Algorithm E.1. Similar to [CLSSW19, Lemma 15], in each iteration, we use $\mathcal{T}_\ell$.FIND$(\cdot)$ to find an out-edge of a_ℓ, taking $\tilde{O}(\beta) = \tilde{O}(\sqrt{r}/d)$ time by Theorem E.4.1. In each iteration, we either increase ℓ and extend the current path by a new element, decrease ℓ and remove one element, or find an (s,t)-path (then remove everything in it), and each element can participate in each of the event at most once. Thus, there are only $O(n)$ iterations, and the total cost of $\mathcal{T}_\ell$.FIND$(\cdot)$ is consequently $\tilde{O}(\frac{n\sqrt{r}}{d})$ by our choice of β. For each of the augmenting path, $\mathcal{T}_\ell$.UPDATE$(\cdot)$ takes $\tilde{O}\left(\frac{|L_\ell| \cdot d}{\sqrt{r}}\right)$ time, contributing to a total running time of $\tilde{O}\left(\frac{(|S'|-|S|) \cdot nd}{\sqrt{r}}\right)$ since L_ℓ's are disjoint.

We then argue the correctness of the algorithm. Observe that at any point in time, $\mathcal{T}_\ell$ is a data structure capable of finding a replacement element with respect to the independent set $S \oplus D_{\ell-1} \oplus D_\ell$, due to the updates that we gave it. This means that the collection $\Pi := (D_1, D_2, \ldots, D_{d_t-1})$ remains an augmenting set in $G(S)$ because $S \oplus (D_{\ell-1} + a_{\ell-1}) \oplus (D_\ell + a_\ell)$ is independent for each ℓ whenever a path is found. As a result, when the algorithm terminates, $S' := S \oplus \Pi$ is indeed a common independent set as guaranteed by Lemma E.3.12.

It remains to show that $d_{G(S')}(s,t) > d_{G(S)}(s,t)$ by arguing that for each a_ℓ not in Π but removed from A_ℓ at time t, there is no shortest augmenting path in $G(S')$ that passes through a_ℓ. This is a direct consequence of Lemma E.5.4 since $\Pi^{(t)}$, the augmenting set obtained at time t, is contained in Π. The fact that $\mathcal{T}^{(t)}_{\ell+1}$.FIND$(a_\ell)$ returns nothing (equivalently, Equation (E.2) is not satisfied) shows that a_ℓ is not on any shortest augmenting path in $G(S')$ since the set X maintained in $\mathcal{T}^{(t)}_{\ell+1}$ (see Theorem E.4.1) is $A_{\ell+1}$ at all time. We remark that x might have an out-edge (with respect to $S \oplus D^{(t)}_\ell \oplus D^{(t+1)}_\ell$) to a removed element with distance $\ell+1$ from s (not in $A_{\ell+1}$), but such an element, by induction, is not on any augmenting path either. □

E.6 Dynamically Maintaining a Basis of a Matroid

In this section, we construct a data structure that allows us to maintain a basis of a matroid in a decremental set. The data structure is used for obtaining an $\tilde{O}_k(n + r\sqrt{r})$ running time for matroid union, but it may be of independent interest as well. Specifically, our data structure has the following guarantees.

Theorem E.6.1. *For a (weighted) matroid $\mathcal{M} = (U, \mathcal{I})$, there exists a data structure supporting the following operations.*

- *INITIALIZE(X): Given a set $X \subseteq U$, initialize the data structure and return a (min-weight) basis S of X in $\tilde{O}(n)$ time.*
- *DELETE(x): Given $x \in X$, remove x from X and return a new (min-weight) basis of X in $\tilde{O}(\sqrt{r})$ time. Specifically, the new basis will contain at most one element (the replacement element of x) not in the old basis, and this procedure returns such an element if any.*

Our data structure for Theorem E.6.1 will consist of two parts. The first part, introduced in Section E.6.1, is a baseline, unsparsified data structure that supports the DELETE operation in $\tilde{O}(\sqrt{n})$ time, and the second one is a sparsification structure which brings the complexity down to $\tilde{O}(\sqrt{r})$, as presented in Section E.6.2.

As hinted by the statement of Theorem E.6.1, to make things simpler, we will assign an arbitrary but unique weight $w(x)$ to each $x \in X$. Now, instead of maintaining an arbitrary basis of X, we maintain the *min-weight* basis instead. The min-weight basis is well-known to be unique (as long as the weights are) and can be obtained greedily as shown in Algorithm E.2 (see, e.g., [Edm71]).

Algorithm E.2: Greedy algorithm for computing the min-weight basis

Input: A set $X \subseteq U$ of size k
Output: The min-weight basis S of X

```
Order X = (x_1, x_2, ..., x_k) so that w(x_1) < w(x_2) < ··· < w(x_k)
S ← ∅
for i ∈ [1, k] do
    if rank(S + x_i) > rank(S) then
        S ← S + x_i
return S
```

Moreover, suppose we remove $x \in S$ from the set X. Then the new min-weight basis is either (i) $S - x + y$ where y is the minimum weight element in $X - x$ that makes $S - x + y$ independent or (ii) simply $S - x$ if such a y does not exist. In case (i), y is called the *replacement* element of x. Note that $w(y) > w(x)$ must hold.

It is useful to note that the S in Line 4 of Algorithm E.2 is interchangeable with $X_{i-1} = \{x_1, \ldots, x_{i-1}\}$, since $\text{span}(X_{i-1}) = \text{span}(S \cap X_{i-1})$, so the sets X_{i-1} and $S \cap X_{i-1}$ have the same rank. In other words, in each iteration i, we can imagine that Algorithm E.2 has chosen every element before x_i.

Observation E.6.2. *In Algorithm E.2, $x_i \in S$ if and only if* $\text{rank}(X_i) > \text{rank}(X_{i-1})$.

E.6.1 Baseline Data Structure

Our baseline data structure supports the operations of Theorem E.6.1, except in time $\tilde{O}(\sqrt{k})$ where $k = |X|$ instead of $\tilde{O}(\sqrt{r})$.

Lemma E.6.3. *For a weighted matroid $\mathcal{M} = (U, \mathcal{I})$, there exists a data structure supporting the following operations.*

- INITIALIZE(X): *Given a set $X \subseteq U$ with $|X| = k$, initialize the data structure and return the min-weight basis S of X in $\tilde{O}(k)$ time.*
- DELETE(x): *Given $x \in X$, remove x from X and return the new min-weight basis of X in $\tilde{O}(\sqrt{k})$ time. Specifically, the new basis will contain at most one element (the replacement element of x) not in the old basis, and this procedure returns such an element if any.*
- INSERT(x): *Given $x \notin X$, add x to X. It's guaranteed that x is not in the min-weight basis of the new X and the size of X does not exceed $2k$.*

Initialization. In the initialization stage, we order X by the weights and split the sequence into $\sqrt{k}$ blocks $X_1, X_2, \ldots, X_{\sqrt{k}}$ from left to right, where each block has roughly the same size $O(\sqrt{k})$. That is, X_1 contains the $\sqrt{k}$ elements with the smallest weights while $X_{\sqrt{k}}$ contains elements with the largest weights. We also compute the basis S of X from left to right as in Algorithm E.2 together with $\sqrt{k}$ query-sets $Q_1, Q_2, \ldots, Q_{\sqrt{k}}$, where $Q_j = \bigcup_{i=1}^{j} X_i$ is the union of the first j blocks. This takes $\tilde{O}(k)$ time in total.

Deletion. For each deletion of x located in the block X_i, we first update the query-sets $Q_i, \ldots, Q_{\sqrt{k}}$ by removing x from them. Let $Q'_1, \ldots, Q'_{\sqrt{k}}$ denote the old query-sets before removing x. If x is not in the basis S we currently maintain, then S remains the min-weight basis of the new X and nothing further needs to be done. Otherwise, we would like to find the min-weight replacement element y of x. We know that such a y, if it exists, can only be located in blocks $X_i, X_{i+1}, \ldots, X_{\sqrt{k}}$. As such, we find the first $j \geq i$ with $\text{rank}(Q_j) = \text{rank}(Q'_j)$ and recompute the portion of S inside X_j. This can be done by running Algorithm E.2 with the initial set S being Q_{i-1}, the union of the first $i-1$ blocks (see Observation E.6.2). Thus, the deletion takes $\tilde{O}(\sqrt{k})$ time.

Insertion. For insertion of x, we simply add x to a block where it belongs (according to $w(x)$) and then update Q_i's appropriately. This takes $\tilde{O}(\sqrt{k})$ as well.

Rebalancing. To maintain an update time of $\tilde{O}(\sqrt{k})$, whenever the size of a block X_i grows larger than $2\sqrt{k}$, we split it into two blocks and recompute Q_i and Q_{i+1}. Similarly, to avoid having too many blocks, whenever the size of a block X_i goes below $\sqrt{k}/2$, we merge it with an adjacent block and remove Q_i. Each of the above operations takes $\tilde{O}(\sqrt{k})$ time, which is subsumed by the cost of an update.

We have shown how to implement each operation of Lemma E.6.3 in its desired running time, and the correctness of the data structure is manifest as we always follow the greedy basis algorithm (Algorithm E.2).

E.6.2 Sparsification

In this section, we prove Theorem E.6.1 by "sparsifying" the input set of the data structure for Lemma E.6.3 in a recursive manner, similar to what [EGIN97] did to improve [Fre85]'s $O(\sqrt{|E|})$ dynamic MST algorithm to $O(\sqrt{|V|})$. The following claim asserts that such sparsification is valid.

Claim E.6.4. *Let S_X and S_Y be the min-weight basis of X and Y, respectively, where $w(x) < w(y)$ holds for each $x \in X$ and $y \in Y$. Then, the min-weight basis of $S_X + S_Y$ is also the min-weight basis of $X + Y$.*

Proof. Consider running the greedy Algorithm E.2 on the set $X + Y$ to obtain the min-weight basis S of it. Clearly, we have $S_X \subseteq S$ since X contains the elements of smaller weights (in fact $S \cap X = S_X$). Assume for contradiction that $S \cap Y \not\subseteq S_Y$, i.e., there exists a $y^* \in S \cap Y$ which does not belong to S_Y. Then, it must be the case that there exists a $y \in S_Y$ with $w(y) > w(y^*)$, as otherwise (i.e., y^* is ordered after everything in S_Y) by Lemma E.3.4 the greedy algorithm stops before seeing y^*. We claim that the greedy algorithm on Y chooses y^* before all such y's, thereby contradicting the fact that S_Y is the min-weight basis of Y. This is true by the diminishing returns property[21] of the rank function: Let Y^* be elements in Y with weights smaller than $w(y^*)$. Since $y^* \in S$, it follows that $\mathrm{rank}(X + Y^* + y^*) > \mathrm{rank}(X + Y^*)$, implying $\mathrm{rank}(Y^* + y^*) > \mathrm{rank}(Y^*)$ and the greedy algorithm run on Y picks y^*. □

We are now ready to present our sparsification data structure.

Proof of Theorem E.6.1. Our data structure is a balanced binary tree where the leaf nodes correspond to elements in X and each internal node corresponds to the set consisting of elements in leaf nodes of this subtree. We will abuse notation and use a node v to also refer to the elements contained in the subtree rooted at v.

[21] The diminishing returns property of submodular functions states that $f(z + X) - f(X) \geq f(z + Y) - f(Y)$ holds for each $Y \subseteq X \subseteq U$ and $z \notin X$.

We first build the binary tree top-down, starting with the root node containing X and recursively splitting the current set into two subsets of roughly the same size and recursing on them.[22] We then build the min-weight basis of each node in a bottom-up manner, starting from the leaves. For each node v with children u_1 and u_2, we initialize the data structure $\mathcal{D}_v$ for Lemma E.6.3 with input set $S_{u_1} + S_{u_2}$, the min-weight basis of u_1 and u_2 which are obtained from $\mathcal{D}_{u_1}$ and $\mathcal{D}_{u_2}$. By Claim E.6.4, the basis $\mathcal{D}_v$ maintains is the min-weight basis of v. Thus, by induction, the basis maintained in the root node is indeed the min-weight basis of the whole set X. The data structure for Lemma E.6.3 takes time near-linear in the size of the input set to construct, and since the sparsified input is a subset of elements in the subtree, the initialization takes time near-linear in the sum of sizes of the subtrees, which is $\tilde{O}(n)$ (indeed, every element occurs in at most $\log n$ nodes).

To delete an element $x \in X$, we first identify the leaf node v_x of the binary tree which corresponds to x. Going upward, for each ancestor p of v_x, we delete x from $\mathcal{D}_p$. If we find a replacement element y for x, we insert y into $\mathcal{D}_q$, where q is p's parent, before proceeding to q (y is not in $\mathcal{D}_p$ so such an insertion is valid by Claim E.6.4). Since x will be removed from $\mathcal{D}_q$ shortly, the input set of $\mathcal{D}_q$ remains the union of the min-weight bases of q's children. This takes $\tilde{O}(\sqrt{r})$ time since $\mathcal{D}_p$ is of size $O(r)$. Inductively, since the min-weight bases of the child nodes are updated, by Claim E.6.4, the min-weight basis of each of the affect nodes (hence the min-weight basis of X) is correctly maintained. □

E.7 Matroid Union

In this section, we present our improved algorithm for matroid union. Our main focus of this algorithm is on optimizing the $O(n\sqrt{r})$ term to $O(r\sqrt{r})$. Thus, for simplicity of presentation, we will treat k as a constant (the dependence on k will be a small polynomial) and express our bounds using the $O_k(\cdot)$ and $\tilde{O}_k(\cdot)$ notation.

Theorem E.7.1. *In the dynamic-rank-oracle model, given k matroids $\mathcal{M}_i = (U_i, \mathcal{I}_i)$ for $1 \le i \le k$, it takes $\tilde{O}_k(n + r\sqrt{r})$ time to find a basis $S \subseteq U_1 \cup \cdots \cup U_k$ of $\mathcal{M} = \mathcal{M}_1 \vee \cdots \vee \mathcal{M}_k$ together with a partition $S_1, \ldots, S_k$ of S in which $S_i \in \mathcal{I}_i$ for each $1 \le i \le k$.*

In Section E.10, we present an optimized (for the parameter k) version of the above algorithm which solves the important special case when all the k matroids are the same—i.e. *k-fold matroid union*—with applications in matroid packing problems. For example, the problem of finding k disjoint spanning trees in a graph falls under this special case. In particular, in Section E.10, we obtain the following Theorem E.7.2, and we discuss some immediate consequences for the *matroid packing*, *matroid covering*, and *k-disjoint spanning trees problems* in Sections E.7.3 and E.7.4.

[22] Note that unlike in Section E.4, we are not building query-sets here.

Theorem E.7.2. *In the dynamic-rank-oracle model, given a matroid $\mathcal{M} = (U, \mathcal{I})$ and an integer k, it takes $\tilde{O}(n + kr\sqrt{\min(n, kr)} + k\min(n, kr))$ time to find the largest $S \subseteq U$ and a partition $S_1, \ldots, S_k$ of S in which $S_i \in \mathcal{I}$ for each $1 \leq i \leq k$.*

The rest of this section will focus on the proof of Theorem E.7.1 (again, where the number of matroids k is treated as a constant). Our algorithm is based on the matroid intersection algorithm in Section E.5, in which we identify and optimize several components that lead to the improved time bound.

E.7.1 Reduction to Matroid Intersection

For completeness, we provide a standard reduction from matroid union to matroid intersection. For an in-depth discussion, see [Sch03, Chapter 42]. Let $\mathcal{M}_i = (U_i, \mathcal{I}_i)$ be the given k matroids and $U = U_1 \cup \cdots \cup U_k$ be the ground set of the matroid union $\mathcal{M} = \mathcal{M}_1 \vee \cdots \vee \mathcal{M}_k$. We first relabel each element in the matroids with an identifier of its matroid, resulting in $\hat{\mathcal{M}}_i = (\hat{U}_i, \mathcal{I}_i)$, where $\hat{U}_i = \{(u, i) \mid u \in U_i\}$. Let $\hat{\mathcal{M}} = (\hat{U}, \hat{\mathcal{I}}) = \hat{\mathcal{M}}_1 \vee \cdots \vee \hat{\mathcal{M}}_k$ be over the ground set $\hat{U} = \hat{U}_1 \sqcup \cdots \sqcup \hat{U}_k$.

In other words, in $\hat{\mathcal{M}}$, we duplicate each element that is shared among multiple matroids into copies that are considered different, effectively making the ground sets of the k matroids disjoint. After this modification, an independent set in $\hat{\mathcal{M}}$ is now simply the union of k independent sets, one from each matroid. However, that might not be what we want since these independent sets may overlap, i.e., contain copies that correspond to the same element. We therefore intersection $\hat{\mathcal{M}}$ with a partition matroid $\mathcal{M}_{\text{part}} = (\hat{U}, \mathcal{I}_{\text{part}})$ given by

$$\mathcal{I}_{\text{part}} = \{S \subseteq \hat{U} \mid |S \cap \{(u, i) \text{ for } 1 \leq i \leq k \mid u \in U_i\}| \leq 1 \text{ holds for each } u \in U\}$$

to restrict different copies of the same element to be chosen at most once. The matroid union problem is thus reducible to the matroid intersection problem in the sense that the intersection of $\hat{\mathcal{M}}$ and $\mathcal{M}_{\text{part}}$ maps exactly to the independent sets of the matroid union $\mathcal{M}$.

Notation-wise, given the above mapping between the two worlds, whenever we write $S \in \mathcal{I}_{\text{part}} \cap \hat{\mathcal{I}}$, a subset set of $\hat{U}$, we will equivalently regard S as a subset of U with an implicit partition $S_1, \ldots, S_k$ where $S_i \in \mathcal{I}_i$.

E.7.2 Specialized Matroid Intersection Algorithm

Given the reduction, to prove Theorem E.7.1, it suffices to compute the intersection of $\hat{\mathcal{M}}$ and $\mathcal{M}_{\text{part}}$ in the claimed time bound. In the following, we will set $\mathcal{M}_1$ to be $\mathcal{M}_{\text{part}}$ and $\mathcal{M}_2$ to be $\hat{\mathcal{M}}$ when talking about exchange graphs and other data structures. Our main goal is to optimize the $O(n\sqrt{r})$ term to $O(r\sqrt{r})$, so it might be more intuitive to think of $r \ll n$. We first show that for an $S \in \mathcal{I}_{\text{part}} \cap \hat{\mathcal{I}}$, the exchange graph $G(S)$ is quite unbalanced in the sense that most elements appear in the first distance layer. In fact, the first distance layer of $G(S)$ contains all duplicates of elements u in U that do not appear in S. This is by definition of $G(S)$

and the fact that $\mathcal{M}_1 = \mathcal{M}_{\text{part}}$ is the partition matroid. In the following, when the context is clear, we let d_t denote the (s,t)-distance of $G(S)$ and $L_1, \ldots, L_{d_t-1}$ denote the distance layers.

Fact E.7.3. *It holds that* $L_1 = \{(u,i) \mid (u,i) \in \hat{U}$ *and* $(u,j) \notin S$ *for any* $1 \leq j \leq k\}$.

Similarly, the odd layers of $G(S)$ (that corresponds to $\bar{S}$) are well-structured in the sense that they consist of elements whose one of the duplicates appears in S. By definition of $G(S)$, we also know that elements in odd layers have only a single in-edge, which is from their corresponding duplicate in S. These elements thus all have the same distance from s.

Fact E.7.4. *It holds that* $L_3 \cup L_5 \cup \cdots \cup L_{d_t-1} = \{(u,i) \mid (u,i) \in \hat{U}$ *and* $(u,j) \in S$ *for some* $i \neq j\}$*, and for each* $(u,i) \in L_3 \cup \cdots \cup L_{d_t-1}$*, we have* $d_{G(S)}(s,(u,i)) = d_{G(S)}(s,(u,j)) + 1$ *where* $(u,j) \in S$.

Union Exchange Graph. Given the above facts, we introduce another notion of exchange graphs which is commonly used for matroid union (see, e.g., [EDVJ68; Cun86]). For the given k matroids $\mathcal{M}_i = (U_i, \mathcal{I}_i)$ and a subset $S \subseteq U$ that can be partitioned into k independent sets $S_1, \ldots, S_k$ with $S_i \in \mathcal{I}_i$, the *union exchange graph* is a directed graph $H(S) = (U \cup \{s,t\}, E)$ with two distinguished vertices $s,t \notin U$ and edge set $E = E_s \cup E_t \cup E_{\text{ex}}$, where

$$\begin{aligned} E_s &= \{(s,u) \mid u \notin S\}, \\ E_t &= \{(u,t) \mid S_i + u \in \mathcal{I}_i \text{ for some } 1 \leq i \leq k\}, \text{ and} \\ E_{\text{ex}} &= \{(u,v) \mid S_i - v + u \in \mathcal{I}_i \text{ where } v \in S_i\}. \end{aligned}$$

We can see that the exchange graph $G(S)$ with respect to $S \in \mathcal{I}_{\text{part}} \cap \hat{\mathcal{I}}$ (as a subset of $\hat{U}$) and the union exchange graph $H(S)$ with respect to $S \subseteq U$ is essentially the same in the sense that $H(S)$ can be obtained from $G(S)$ by contracting all copies of the same element in the first layers and skipping all other odd layers. In particular, for each $(u,i) \in S$, in $G(S)$, there might be a direct edge from (u,i) to (u,j) and an edge from (u,j) to (v,j), where $(v,j) \in S_j$ and $S_j - v + u \in \mathcal{I}_j$. Correspondingly, in $H(S)$, we skip the intermediate vertex (u,j) and meld the above two edges as one direct edge from $u \in S_i$ to $v \in S_j$. We also merge all edges from s to some (u,i) of the same u in the first layer to a single edge from s to u (Fact E.7.3). This simplification does not impact the distance layers of $H(S)$ since all such (u,j) have the same distance from s (Fact E.7.4).

From now on, for simplicity, our algorithms will run on the union exchange graphs $H(S)$, i.e., we will perform blocking-flow computation and augment S along paths in $H(S)$. On the other hand, to not repeat and specialize all the lemmas to

the case of union exchange graphs, proofs and correctness will be argued implicitly in the perspective of the exchange graph $G(S)$ for matroid intersection. For instance, for $P = (s, a_1, \ldots, a_{d_t-1}, t)$ a shortest (s,t)-path in $H(S)$, "augmenting S along P" means moving a_i to the independent set that originally contains a_{i+1} for each $i \geq 1$, and thus effectively enlarge the size of S by one via putting a_1 in it.[23] One can verify that this is indeed what happens if we map P back to a path P' in $G(S)$, and then perform the augmentation of S (as a subset of $\hat{U}$) along P'.

Our main idea to speed up the matroid union algorithm to $\tilde{O}_k(r\sqrt{r})$ (instead of $\tilde{O}_k(n\sqrt{r})$) is to "sparsify" the first layer of $H(S)$ by only considering a subset of elements contained in some basis. We formalize this in the following Lemmas E.7.5 and E.7.6 together with Algorithms E.3 and E.4.

Lemma E.7.5. *Given $S \in \mathcal{I}_{part} \cap \hat{\mathcal{I}}$ and k bases $\{B_i\}_{i=1}^k$ of $U_i \setminus S$, it takes $\tilde{O}_k(r)$ time to construct the distance layers $L_2, \ldots, L_{d_t-1}$ of $H(S)$.*

Note that we know exactly what elements are in the first distance layer, so computing $L_2, \ldots, L_{d_t-1}$ suffices.

Algorithm E.3: BFS in a union exchange graph

Input: $S \subseteq U$ which partitions into $S_1, \ldots, S_k$ of independent sets and k bases $\{B_i\}_{i=1}^k$ of $U_i \setminus S$

Output: The (s,u)-distance $d(u)$ in $H(S)$ for each $u \in S \cup \{t\}$

```
queue ← B_1 ∪ ··· ∪ B_k
d(u) ← ∞ for each u ∈ S ∪ {t}, and d(u) ← 1 for each u ∈ B_1 ∪ ··· ∪ B_k
T_i ← INITIALIZE(M_i, S_i, S_i) (Theorem E.4.1 with β = 1)
while queue ≠ ∅ do
    u ← queue.POP()
    for i ∈ {1, 2, ..., k} where u ∈ U_i and u ∉ S_i do
        while v := T_i.FIND(u) ≠ ⊥ do
            d(v) ← d(u) + 1 and queue.PUSH(v)
            T_i.DELETE(v)
        if S_i + u ∈ I and d(t) = ∞ then d(t) ← d(u) + 1
return d(u) for each u ∈ S ∪ {t}
```

Proof. The algorithm is presented as Algorithm E.3, and it is essentially a breadth-first-search (BFS) starting from $B_1 \cup \cdots \cup B_k$ instead of s. Out-edges in $H(S)$ are explored via k binary search trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ of Theorem E.4.1, one for each matroid $\mathcal{M}_i$ and independent set S_i. Let's analyze the running time first.

[23] One can show that the matroid union $\mathcal{M}$ is a matroid [Sch03, Chapter 42]. As such, a basis can be obtained by trying to include each element into S. From the union exchange graph perspective, the independence test of $S + x$ corresponds to asking whether "there is a path in $H(S)$ from x to t".

Building $\mathcal{T}_i$ takes a total of $\tilde{O}(|S|) = \tilde{O}_k(r)$ time. Exploring the graph takes $\tilde{O}(|S \cup B_1 \cup \cdots \cup B_k| \cdot k) = \tilde{O}_k(r)$ time in total since each element in S is found at most once by $\mathcal{T}_i$.FIND$(\cdot)$ because S_i's are disjoint, and we also spend $O(k)$ time for each element in $S \cup B_1 \cup \cdots \cup B_k$ iterating over $\mathcal{T}_i$.

It remains to show that starting from $B_1 \cup \cdots \cup B_k$ instead of $U \setminus S$ does not affect the correctness of the BFS. For this, it suffices to show that we successfully compute $d(u)$ for all $u \in S$ with distance 2 from s. By definition, $u \in S_i$ is of distance 2 from s if and only if there exists an $x \in U_i \setminus S$ such that $S_i - u + x \in \mathcal{I}_i$. This is equivalent to $\text{rank}_i(S_i - u + (U_i \setminus S)) > \text{rank}_i(S_i)$ by Observation E.4.3. But then by Lemma E.3.4, we have $\text{rank}_i(S_i - u + (U_i \setminus S)) = \text{rank}_i(S_i - u + B_i)$, and so such an x exists in B_i as well. This concludes the proof of Lemma E.7.5. □

Lemma E.7.6. *Given an $S \in \mathcal{I}_{part} \cap \hat{\mathcal{I}}$ with $d_{H(S)}(s,t) = d_t$ together with data structures $\mathcal{D}_i$ of Theorem E.6.1 that maintains a basis of $U_i \setminus S$ for each $1 \le i \le k$, it takes $\tilde{O}_k(r + \frac{r\sqrt{r}}{d_t} + (|S'| - |S|) \cdot d_t\sqrt{r})$ time to obtain an $S' \in \mathcal{I}_{part} \cap \hat{\mathcal{I}}$ with $d_H(S')(s,t) > d_t$, with an additional guarantee that $\mathcal{D}_i$ now maintains a basis of $U_i \setminus S'$ for each $1 \le i \le k$.*

Proof of Lemma E.7.6. Our blocking-flow algorithm for matroid union is presented as Algorithm E.4. As it is equivalent to Algorithm E.1 running on $G(S)$ except that the first layer $L_1 := B_1 \cup \cdots \cup B_k$ is now only a subset (which is updated after each augmentation) of $U \setminus S$, we skip most parts of the proof and focus on discussing this difference. That is, we need to show that if A_1 becomes empty, then there is no augmenting path of length d_t in $H(S')$ anymore. Given how A_1 and B_i's are maintained and Lemma E.5.4 (note that the set X maintained in $\mathcal{T}_\ell^{(i)}$ is always $A_\ell \cap S_i$ with respect to the current S_i and thus it lets us explore out-edges to $A_\ell \cap S_i$ satisfying Equation (E.2)), A_1 is always the subset of $B_1 \cup \cdots \cup B_k$ consisting of elements that still potentially admits augmenting path of length d_t in $H(S')$ through them. That means if $A_1 = \emptyset$, then there is no augmenting set of length d_t in $G(S')$, that starts from some $b \in B_1 \cup \cdots \cup B_k$. This would imply that there is no such path even if we start from $x \in (U_i \setminus S) \setminus D_1$ as B_i is a basis of it: if $S_i + x - y \in \mathcal{I}$ for some $x \in (U_i \setminus S) \setminus D_1$ and $y \in S_i$, then there is a $b \in B$ with $S + b - y \in \mathcal{I}$, and thus a path starting from x can be converted into a path starting from b. On the other hand, all elements in D_1 are not on a such path by Lemma E.3.9 either. This shows that indeed $d_{H(S')}(s,t) > d_{H(S)}(s,t)$.

The guarantee that $\mathcal{D}_i$ now operates on $U_i \setminus S'$ is clear: Augmenting along $P = (s, a_1, \ldots, a_{d_t-1}, t)$ corresponds to adding a_1 into S, and since we call $\mathcal{D}_i$.DELETE(a_1) in Line 22 after each such augmentation, $\mathcal{D}_i$ indeed stays up-to-date.

It remains to analyze the running time of Algorithm E.4. Computing distance layers with Lemma E.7.5 takes $\tilde{O}_k(r)$ time. The number of elements that have ever been in some A_i is $O_k(r + |S'| - |S|)$ since (i) $L_2 \cup \cdots \cup L_{d_t-1}$ has size $O_k(r)$, (ii) the initial basis B_i of $U_i \setminus S$ for each $1 \le i \le k$ has total size $O_k(r)$, and (iii) each of the $|S'| - |S|$ augmentations adds at most $O_k(1)$ elements to A_1. Similar

Algorithm E.4: Blocking flow in a union exchange graph

Input: $S \subseteq U$ which partitions into $S_1, \ldots, S_k$ of independent sets and a dynamic-basis data structure $\mathcal{D}_i$ of $U_i \setminus S$ for each $1 \leq i \leq k$
Output: $S' \in \mathcal{I}_{\text{part}} \cap \mathcal{I}_k$ with $d_{H(S')}(s,t) > d_{H(S)}(s,t)$
Guarantee: $\mathcal{D}_i$ maintains a basis of $U_i \setminus S'$ at the end of the algorithm for each $1 \leq i \leq k$

Build the distance layers $L_2, \ldots, L_{d_t-1}$ of $H(S)$ with Lemma E.7.5
$L_0 \leftarrow \{s\}$ and $L_{d_t} \leftarrow \{t\}$
$B_i \leftarrow$ the basis maintained by $\mathcal{D}_i$ and $L_1 \leftarrow B_1 \cup \cdots \cup B_k$
$A_\ell \leftarrow L_\ell$ for each $0 \leq \ell \leq d_t$
$\mathcal{T}_\ell^{(i)} \leftarrow$ INITIALIZE$(\mathcal{M}_i, S_i, Q_{S_i}, A_\ell \cap S_i)$ for each $2 \leq \ell < d_t$ and $1 \leq i \leq k$ (Theorem E.4.1 with $\beta = \sqrt{r}/d_t$)
$D_\ell \leftarrow \emptyset$ for each $1 \leq \ell < d_t$
$\ell \leftarrow 0$ and $a_0 \leftarrow s$
while $\ell \geq 0$ **do**
  if $\ell < d_t$ **then**
    if $A_\ell = \emptyset$ **then break**
    if $\ell = 0$ **then** Find an $a_{\ell+1} := \mathcal{T}_{\ell+1}^{(i)}$.FIND$(a_\ell) \neq \perp$ for some $1 \leq i \leq k$
    else $a_{\ell+1} \leftarrow$ an arbitrary element in A_1
    if *such an* $a_{\ell+1}$ *does not exist* **then**
      if $\ell \geq 2$ **then** $\mathcal{T}_\ell^{(j)}$.DELETE(a_ℓ) where $a_\ell \in S_j$
      $A_\ell \leftarrow A_\ell - a_\ell$ and $\ell \leftarrow \ell - 1$
    else
      $\ell \leftarrow \ell + 1$
  else
    // Found augmenting path $a_1, a_2, \ldots a_\ell$
    $D_1 \leftarrow D_1 + a_1$ and $A_1 \leftarrow A_1 - a_1$
    for $i \in \{1, 2, \ldots, k\}$ *where* $a_1 \in U_i$ **do**
      $B_i \leftarrow B_i - a_1$
      if $\mathcal{D}_i$.DELETE(a_1) *returns a replacement* x **then**
        $B_i \leftarrow B_i + x$ and $A_1 \leftarrow A_1 \cup \{x\}$
    for $i \in \{2, \ldots, d_t - 1\}$ **do**
      $D_i \leftarrow D_i + a_i$ and $A_i \leftarrow A_i - a_i$
      $\mathcal{T}_i^{(j)}$.DELETE(a_i) and $\mathcal{T}_i^{(j)}$.UPDATE$(\{a_{i-1}, a_i\})$ where $a_i \in S_j$
    Augment S along $P = (s, a_1, \ldots, a_{d_t-1}, t)$
    $\ell \leftarrow 0$
return S

to Lemma E.5.3, this means that there are at most $O_k(r)$ iterations, each taking $O_k(\frac{\sqrt{r}}{d_t})$ time in $\mathcal{T}^{(i)}_{\ell+1}$.FIND$(\cdot)$ with our choice of β. The algorithm found $|S'| - |S|$ augmenting paths, taking $\tilde{O}_k(d_t\sqrt{r} \cdot (|S'| - |S|))$ time in total to update the binary search trees. Also, for each such augmentation, we need $\tilde{O}_k(\sqrt{r})$ time to update the basis B_i for all $1 \le i \le k$, which is subsumed by the cost of updating $\mathcal{T}^{(j)}_i$. These components sum up the total running time of

$$\tilde{O}_k\left(r + \frac{r\sqrt{r}}{d_t} + (|S'| - |S|) \cdot d_t\sqrt{r}\right).$$

□

Theorem E.7.1 now follows easily.

Proof of Theorem E.7.1. We initialize the dynamic-basis data structure $\mathcal{D}_i$ of Theorem E.6.1 on U_i for each of the matroid $\mathcal{M}_i$. We then run Lemma E.7.6 for at most $\sqrt{r}$ iterations with $\{\mathcal{D}_i\}_{i=1}^k$ until $d_{H(S)}(s,t) \ge \sqrt{r}$ and get an $S \in \mathcal{I}_{\text{part}} \cap \hat{\mathcal{I}}$ with $\mathcal{D}_i$ now operating on $U_i \setminus S$ for each $1 \le i \le k$. This takes

$$\tilde{O}_k\left(r\sqrt{r} + \sum_{d=1}^{\sqrt{r}} \frac{r\sqrt{r}}{d} + \sum_{d=1}^{\sqrt{r}} d \cdot (|S_d| - |S_{d-1}|)\right) = \tilde{O}_k(r\sqrt{r})$$

time. By Lemma E.3.7, S is $O_k(\sqrt{r})$ steps away from being optimal, and thus we find the remaining augmenting paths one at a time using Lemma E.7.5 in $\tilde{O}_k(r\sqrt{r})$ time in total. Note that since a single augmentation corresponds to adding an element to S (hence removing it from $U \setminus S$), we can maintain the basis of $U_i \setminus S$ that Lemma E.7.5 needs in $\tilde{O}_k(\sqrt{r} \cdot \sqrt{r})$ total update time, which is subsumed by other parts of the algorithm. □

E.7.3 Matroid Packing and Covering

A direct consequence of our matroid union algorithm (Theorem E.7.2 in particular) is that we can solve the following packing and covering problem efficiently. As a reminder, the exact dependence on k of our algorithm is $\tilde{O}(n + kr\sqrt{\min(n, kr)} + k\min(n, kr))$ by Theorem E.7.2.

Corollary E.7.7 (Packing)**.** *For a matroid $\mathcal{M} = (U, \mathcal{I})$, it takes $\tilde{O}(n\sqrt{n} + \frac{n^2}{r})$ time to find the largest integer k and a collection of disjoint subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of U such that S_i is a basis for each $1 \le i \le k$ under the dynamic-rank-query model.*

Proof. It's obvious that $k \le \frac{n}{r}$ holds. We do a binary search of k in the range $[0, \frac{n}{r}]$, and for each k, we can determine the largest subset S of U which can be partitioned into k disjoint independent sets by Theorem E.7.2. If $|S| = kr$, then it means that there are at least k disjoint bases. Otherwise, there are less than k disjoint bases. The running time is $\tilde{O}(n\sqrt{n} + \frac{n^2}{r})$. □

Corollary E.7.8 (Covering)**.** *For a matroid $\mathcal{M} = (U, \mathcal{I})$, it takes $\tilde{O}(\alpha r\sqrt{n} + \alpha n)$ time to find the smallest integer α and a partition $\mathcal{S} = \{S_1, S_2, \ldots, S_\alpha\}$ of U such that $S_i \in \mathcal{I}$ holds for each $1 \leq i \leq \alpha$ under the dynamic-rank-query model.*

Proof. We first obtain a 2-approximation α' of α (i.e., $\alpha \leq \alpha' \leq 2\alpha$) by enumerating powers of 2, running Theorem E.7.2 with $k = 2^i$, and checking if the returned S has size n: If $|S| = n$, then we know 2^i independent sets suffice to cover U. Note that the enumeration stops whenever we found a suitable value of α'. The exact value of α can then be found by a binary search in $[\frac{\alpha'}{2}, \alpha']$. This takes $\tilde{O}(\alpha r\sqrt{n} + \alpha n)$ (note that $\alpha r \geq n$ must hold). □

E.7.4 Application: Spanning Tree Packing

We demonstrate the applicability of our techniques by deriving an $\tilde{O}(|E| + (k|V|)^{3/2})$ algorithm for the k disjoint spanning tree problem in a black-box manner. This improves Gabow's specialized $\tilde{O}(k^{3/2}|V|\sqrt{|E|})$ algorithm [GW88]. Since all applications of our algorithms follow the same reduction, we only go through it once here. Refer to Section E.11 for other applications of both our matroid union and matroid intersection algorithms.

Theorem E.7.9. *Given an undirected graph $G = (V, E)$, it takes $\tilde{O}(|E| + (k|V|)^{3/2})$ time to find k edge-disjoint spanning trees in G or determine that such spanning trees do not exist with high probability*[24]*.*

Proof. By Theorem E.7.2, it suffices to provide a data structure that supports the three dynamic-oracle operations (Definition E.1.2) in polylog($|V|$) time. Our black-box reduction makes use of the worst-case connectivity data structure of [KKM13; GKKT15], which can be adapted to in $O(\text{polylog}(|V|))$ update time maintain the rank of a set of edges (see Section E.11.2 for a discussion on how this can be done).

Let $\mathcal{M}_G$ be the graphic matroid with respect to $G = (V, E)$. G admits k edge-disjoint spanning trees if and only if $\mathcal{M}_G$ admits k disjoint bases. The theorem now follows from Theorem E.7.2 with $n = |E|$ and $r = |V| - 1$ since Theorem E.7.2 returns a union of k disjoint bases if they exist (we note that $k \leq |E|/(|V| - 1) \leq O(|V|)$, and hence the $O(k^2 r)$ term is dominated by the $O((kr)^{3/2})$ term). □

E.8 Super-Linear Query Lower Bounds

Lower bounds for matroid intersection have been notoriously difficult to prove. The current highest lower bound is due to Harvey [Har08] which says that $(\log_2 3)n - o(n)$ queries are necessary for any deterministic independence-query algorithm solving matroid intersection. Obtaining an $\omega(n)$ lower bound has been called a challenging open question [CLSSW19].

[24] We use *with high probability* to denote with probability at least $1 - |V|^{-c}$ for an arbitrarily large constant c.

In this section, we show the first super-linear query lower bound for matroid intersection, both in our *new dynamic-rank-oracle* model (Definition E.1.2), and also for the *traditional independence-oracle* model, thus answering the above-mentioned open question and improving on the bounds of [Har08]. We obtain our lower bounds by studying the communication complexity for matroid intersection.

Theorem E.8.1. *If Alice is given a matroid $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and Bob a matroid $\mathcal{M}_2 = (U, \mathcal{I}_2)$, any deterministic communication protocol needs $\Omega(n \log n)$ bits of communication to solve the matroid intersection problem.*

The communication lower bound of Theorem E.8.1 implies a similar lower bound for the number of independence queries needed. We argue that any independence-query algorithm can be simulated by Alice and Bob in the communication setting by exchanging a single bit per query asked. Whenever they want to ask an independence query "Is $S \in \mathcal{I}_i$?", Alice or Bob will check this locally and share the answer with the other party by sending one bit of communication.

Unfortunately, this argument does not extend to the traditional rank-oracle model (since each rank query can in fact reveal $\Theta(\log n)$ bits of information, which need to be sent to the other party). However, for the new *dynamic*-rank-oracle model, the $\Omega(n \log n)$ lower bound holds as now each new query only reveals constant bits of information: either the rank remains the same, increases by one, or decreases by one (and Alice or Bob can send which is the case to the other party with a constant number of bits). Our discussion proves the following corollaries, given Theorem E.8.1.

Corollary E.8.2. *Any deterministic (traditional) independence-query algorithm solving matroid intersection requires $\Omega(n \log n)$ queries.*

Corollary E.8.3. *Any deterministic dynamic-rank-query algorithm solving matroid intersection requires $\Omega(n \log n)$ queries.*

Remark E.8.4. We note that our lower bounds are also valid for the *matroid union* problem, due to the standard reductions[25] between matroid intersection and union.

E.8.1 Communication Setting

We study the following communication game which we call Matroid-Intersection-with-Candidate. Alice and Bob are given matroids $\mathcal{M}_1 = (U, \mathcal{I}_1)$ respectively $\mathcal{M}_2 = (U, \mathcal{I}_2)$. Suppose they are also both given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, and they wish to determine whether S is a maximum-cardinality

[25] See Section E.7.1 for a reduction from matroid union to matroid intersection. To reduce from matroid intersection to matroid union, consider $\mathcal{M} = \mathcal{M}_1 \vee \mathcal{M}_2^*$, where $\mathcal{M}_2^*$ is the *dual matroid* of $\mathcal{M}_2$ ($S \subseteq U$ is independent in $\mathcal{M}_2^*$ if and only if $U - S$ contains a basis). It's easy to show that the basis B of $\mathcal{M}$ in $\mathcal{M}_1$ will be of the form $B = S \cup (U \setminus R)$, where S is the solution to the intersection between $\mathcal{M}_1$ and $\mathcal{M}_2$ and R is an arbitrary basis of $\mathcal{M}_2$ that contains S.

independent set. Clearly Matroid-Intersection-with-Candidate is an easier version of the matroid intersection problem, as Alice and Bob can just ignore the candidate S.

Our idea is that in order to solve Matroid-Intersection-with-Candidate, Alice and Bob need to determine if there exists an *augmenting path*—that is an (s,t)-path—in the *exchange graph* $G(S)$ (see Definition E.3.5 and Lemma E.3.6). It is known that (s,t)-connectivity in a graph requires $\Omega(n \log n)$ bits of communication (Lemma E.8.8, [HMT88]). Using *strict gammoids* as our matroids, we argue that we can choose exactly how the underlying exchange graph looks like, and hence that matroid intersection admits the same lower bound.

Definition E.8.5 (Strict Gammoid, see [Per68; Mas72])**.** Let $H = (V, E)$ be a directed graph and $X \subseteq V$ a subset of vertices. Then (H, X) defines a matroid $\mathcal{M} = (V, \mathcal{I})$ called a *strict gammoid*, where a set of vertices $Y \subseteq V$ is independent if and only if there exists a set of vertex-disjoint directed paths (some of which might just consist of single vertices) in H whose starting points all belong to X and whose ending points are exactly Y.

Claim E.8.6. *Suppose $G = (L, R, E)$ is a directed bipartite graph and $a, b \in R$ are two unique vertices such that a has zero in-degree and b has zero out-degree. Then there exist two matroids $\mathcal{M}_1, \mathcal{M}_2$ over the ground set $L \cup R$ such that L is independent in both matroids and the* exchange graph *$G(L)$ is exactly G plus two extra vertices (s and t) and two extra edges ($s \to a$ and $b \to t$).*

Proof. Let $F_1 = \{(u,v) | (u,v) \in E, u \in L, v \in R\}$ be the directed edges from L to R in G, and $F_2 = \{(u,v) | (v,u) \in E, u \in L, v \in R\}$ be the (reversed) directed edges from R to L in G. Also let $H_1 = (L \cup R, F_1)$ and $H_2 = (L \cup R, F_2)$ be the directed graphs with these edges respectively.

We let $\mathcal{M}_1$, respectively $\mathcal{M}_2$, be the strict gammoids defined by $(H_1, L + a)$ respectively $(H_2, L+b)$. Now L is independent in both matroids. It is straightforward to verify that the exchange graph $G(L)$ is exactly as described in the claim. We certify that this is the case for the edges defined by $\mathcal{M}_1$ ($\mathcal{M}_2$ is similar):

1. $G(L)$ will have an edge from s to a, since $L + a$ is independent in $\mathcal{M}_1$. Additionally note that a has in-degree zero in G (and hence is an isolated vertex in H_1).

2. For any $x \in L, y \in R$, the edge (x, y) exists in $G(L)$ if and only if $L - x + y$ is independent in $\mathcal{M}_1$. By definition this is if and only if there exists a vertex-disjoint path starting from L and ending to $L - x + y$ in H_1, or equivalently if the edge (x, y) exists in H_1 (indeed, all vertices in $L - x$ must be both starts and ends of paths, so the path to y must have started in x). □

We now proceed to reduce an instance of (s,t)-connectivity to that of Matroid-Intersection-with-Candidate, which concludes the proof of Theorem E.8.1.

Definition E.8.7 ((s,t)-connectivity)**.** Suppose $G = (V, E_A \cup E_B)$ is an undirected graphs on $n = |V|$ vertices, where Alice knows edges E_A and Bob knows edges E_B. They are also both given vertices s and t, and want to determine if s and t are connected in G.

Lemma E.8.8 ([HMT88])**.** *The deterministic communication complexity of* (s,t)-connectivity *is* $\Omega(n \log n)$.

Proof of Theorem E.8.1. We show that an instance of (s,t)-connectivity can be converted to an instance of Matroid-Intersection-with-Candidate of roughly the same size. Suppose the symbols are defined as in Definition E.8.7. Let $\bar{V} = \{\bar{v} : v \in V\}$ be a copy of V. We construct a directed bipartite graph $G' = (V, \bar{V}, E'_A \cup E'_B)$ as follows:

- $(v, \bar{v}) \in E'_A$ for all $v \in V$.
- $(\bar{v}, v) \in E'_B$ for all $v \in V$.
- $(v, \bar{u}), (u, \bar{v}) \in E'_A$ for all $\{u, v\} \in E_A$.
- $(\bar{v}, u), (\bar{u}, v) \in E'_A$ for all $\{u, v\} \in E_B$.
- No other edges exist.

Alice knows E'_A, and Bob knows E'_B. G' has $2n$ vertices and $2n + 2m$ edges.

Now let G'' be G' but removing all incoming edges from $\bar{s}$ and all outgoing edges from $\bar{t}$, in order to apply Claim E.8.6 on G'' with $a = \bar{s}$ and $b = \bar{t}$. Say we get matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. Note that Alice knows $\mathcal{M}_1$ and Bob knows $\mathcal{M}_2$ by construction.

Now s and t are connected in G if and only if there is a directed $(\bar{s}, \bar{t})$-path in G''. This happens if and only if V is not a maximum-cardinality common independent set of $\mathcal{M}_1$ and $\mathcal{M}_2$ (i.e. in the case we found an augmenting path for V).

Hence if there is a (deterministic) communication protocol for matroid intersection using c bits of communication, there is also one for (s,t)-connectivity using $O(c)$ bits of communication. Lemma E.8.8 then implies the $\Omega(n \log n)$ communication lower bound for matroid intersection. □

E.9 Open Problems

Our dynamic-oracle model opens up a new path to achieve fast algorithms for many problems at once, where the ultimate goal is to achieve near-linear time and dynamic-rank-query complexities. This would imply near-linear time algorithms for many fundamental problems. We envision reaching this goal via a research program where the studies of algorithms and lower bounds in our and the traditional models complement each other. In particular, a major open problem is to improve our algorithms further, which would imply improved algorithms for many problems

simultaneously. A major step towards this goal is improved algorithms in the traditional model, which would already be a breakthrough. Moreover, failed lower bound attempts might lead to new algorithmic insights and vice versa, and we leave improving our lower bounds as another major open problem. We believe that the communication complexity of graph and matroid problems is an important component in this study since it plays a main role in our lower bound argument. Recently the communication and some query complexities of bipartite matching and related problems were resolved in [BBEMN22]. How about the communication and query complexities of dynamic-oracle matroid problems and their special cases such as colorful spanning trees? It is also fruitful to resolve some special cases as the solutions may shed more light on how to solve matroid problems in our model. Below are some examples.

- **Disjoint Spanning Trees.** Can we find k edge-disjoint spanning trees in an undirected graph in near-linear time for constant k, or even do so for the case of $k = 2$ (which already has application in the Shannon Switching Game)? Our new $\tilde{O}(|E| + |V|\sqrt{|V|})$-time algorithm shows that it is possible for sufficiently dense graphs. For the closely related problem of finding k edge-disjoint arborescences (rooted *directed* spanning trees) in a directed graph, the case of $k = 2$ has long been settled by Tarjan's linear time algorithm [Tar76], and the case of constant k has also been resolved by [BHKP08]. It is a very interesting question whether the directed case is actually computationally easier than the undirected case or not.

- **Colorful Spanning Tree.** This problem generalizes the maximum bipartite matching problem, among others. Given the recent advances in max-flow algorithms which are heavily based on continuous optimization techniques, bipartite matching can now be solved in almost-linear time [CKLPGS22] in general and nearly linear time for dense input graphs [BLNPSSSW20]. It is very unclear if continuous optimization can be used for colorful spanning tree since its linear program has exponentially many constraints. This reflects the general challenge of using continuous optimization to solve matroid problems and many of their special cases. Thus, improving Hopcroft-Karp's $O(|E|\sqrt{|V|})$ runtime [HK73] (which is matched by our dynamic-oracle matroid algorithm) may shed some light on either how to use continuous optimization for these problems or how combinatorial algorithms can break this runtime barrier for colorful spanning tree, bipartite matching, and matroid problems.

Other Problems with Dynamic Oracles. It also makes sense to define dynamic oracles for problems like submodular function minimization (SFM), which asks to find the minimizer of a submodular function given an evaluation oracle. In this regime, similar to matroid intersection, we want to limit the symmetric difference from the current evaluation query to the previous ones. We believe that the recent algorithms for submodular function minimization based on convex optimization and

cutting-plane methods, particularly the work of [LSW15; JLSW20], can be adapted to the dynamic-oracle setting. However, we are not aware of any applications of these dynamic-oracle algorithms. The first step is thus improving the best bounds in the traditional oracle model. The special case of the cut-query setting [RSW18; MN20; LSZ21; LLSZ21; AEGLMN22] is also very interesting; we leave getting algorithms for min (s,t)-cut [CKLPGS22] and directed global mincut [CLNPSQ21] with near-linear time and dynamic-query complexity as major open problems.[26] Another interesting direction is the quantum setting. For example, can one define the notion of dynamic quantum cut query so that the quantum cut-query algorithm of [AEGLMN22] can imply a non-trivial quantum global mincut algorithm?

Improved Lower Bounds. Obtaining improved lower bounds for matroid intersection is also an important open problem. Getting $\Omega(n \log n)$ lower bound for traditional rank-query matroid intersection algorithms is particularly interesting since it would subsume our $\Omega(n \log n)$ lower bounds (traditional rank-query lower bound implies independence-query and dynamic-rank-query lower bounds) and the $\Omega(n \log n)$ SFM lower bound of [CGJS22]. For the latter, [CGJS22] showed an $\Omega(n \log n)$ lower bound for SFM against *strongly*-polynomial time algorithms. Since SFM generalizes matroid intersection in the traditional rank-oracle model (i.e., a rank query of a matroid corresponds to an evaluation of the submodular function), getting the same lower bound for traditional rank-query matroid intersection algorithms would further strengthen the result of [CGJS22] to hold against *weakly*-polynomial time algorithms.

Additionally, achieving a *truly* super-linear lower bound (i.e. an $n^{1+\Omega(1)}$ bound) for any of the above problems is extremely interesting.

APPENDIX

E.10 k-Fold Matroid Union

In this section, we study the special case of matroid union where we take the k-fold union of the same matroid. That is, a basis of the k-fold union of $\mathcal{M} = (U, \mathcal{I})$ is the largest subset $S \subseteq U$ which can be partitioned into k disjoint independent sets $S_1, \ldots, S_k$ of $\mathcal{I}$. Many of the prominent applications of matroid union fall into this special case, particularly the k-disjoint spanning tree problem. As a result, here we show an optimized version of the algorithm presented in Section E.7 with better and explicit dependence on k that works in this regime.

Theorem E.7.2. *In the dynamic-rank-oracle model, given a matroid $\mathcal{M} = (U, \mathcal{I})$ and an integer k, it takes $\tilde{O}(n + kr\sqrt{\min(n, kr)} + k\min(n, kr))$ time to find the largest $S \subseteq U$ and a partition $S_1, \ldots, S_k$ of S in which $S_i \in \mathcal{I}$ for each $1 \leq i \leq k$.*

[26] Adapting the cut-query algorithm of [MN20] to work with dynamic cut oracles and, even better, with a parallel algorithm [AB21; LMN21], is also open; though, we suspect that these are not hard.

Note that in the breadth-first-search and blocking-flow algorithms in Section E.7, there is an $O(k)$ overhead where we have to spend $O(k)$ time iterating through the k binary search trees in order to explore the out-neighbors of the $O(kr)$ elements. Our goal in this section is thus to show that it is possible to further "sparsify" the exchange graphs to contain essentially only a basis, hence reducing its size from $\Theta(kr)$ to $O(r)$. We start with a slight modification to the BFS Algorithm E.3 which reduces the running time by a factor of $O(k)$. The idea is that if we visit an element u in the BFS which does not increase the rank of all visited elements so far, we can skip searching out-edges from u. Indeed, if (u, v) is an edge of the exchange graph, then there must have been some element u' visited earlier in the BFS which also has the edge (u', v).

Algorithm E.5: BFS in a k-fold union exchange graph

Input: $S \subseteq U$ with partition $S_1, \ldots, S_k$ of independent sets and a basis B of $U \setminus S$

Output: The (s, v)-distance $d(v)$ in $H(S)$ for each $v \in S \cup \{t\}$

```
queue ← B and R ← ∅
d(v) ← ∞ for each v ∈ S ∪ {t}, and d(v) ← 1 for each v ∈ B
T_i ← INITIALIZE(M, S_i, S_i) (Theorem E.4.1 with β = 1)
while queue ≠ ∅ do
    u ← queue.POP()
    if R + u ∈ I then
        for i ∈ {1, 2, ..., k} do
            while v := T_i.FIND(u) ≠ ⊥ do
                d(v) ← d(u) + 1 and queue.PUSH(v)
                T_i.DELETE(v)
            if S_i + u ∈ I and d(t) = ∞ then d(t) ← d(u) + 1
        R ← R + u
return d(v) for each v ∈ S ∪ {t}.
```

Lemma E.10.1. *Given $S \in \mathcal{I}_{part} \cap \hat{\mathcal{I}}$ and a basis B of $U \setminus S$, it takes $\tilde{O}(kr)$ time to construct the distance layers $L_2, \ldots, L_{d_t - 1}$ of $H(S)$.*

Proof. The algorithm is presented as Algorithm E.5. It performs a BFS from a basis B of the first layer and only explores out-edges from the first basis R it found. It takes (i) $\tilde{O}(|S|)$ time to construct the $\mathcal{T}_i$'s, (ii) $\tilde{O}(1)$ time to discover each of the $O(kr)$ element, and (iii) an additional $\tilde{O}(k \cdot |R|)$ time to iterate through all k binary search trees $\mathcal{T}_i$'s for each $u \in R$. The total running time is thus bounded by $\tilde{O}(kr)$.

We have shown in Lemma E.7.5 that it is feasible to replace $U \setminus S$ with simply B. It remains to show that exploring only the out-neighbors of $u \in R$ does not affect the correctness. Consider a $v \in S \setminus R$ (we know that $B \subseteq R$ so it suffices to consider

elements in S) with an out-neighbor $x \in S_i$, i.e., $S_i - x + v \in \mathcal{I}$. It then follows that $\text{rank}((S_i - x) + R_v) = \text{rank}((S_i - x) + (R_v + v)) \geq \text{rank}(S_i)$ by Observation E.4.3 and Lemma E.3.4, where R_v is the set R when v is popped out of the queue (in other words, $R_v + v \notin \mathcal{I}$). This implies that there is a $u \in R_v$ which is visited before v that also has out-neighbor x. The modification is therefore correct. □

Our blocking-flow algorithm for k-fold matroid union is presented as Algorithm E.6. It's essentially a specialization of Algorithm E.4 to the case where all the k matroids are the same, except that we skip exploring the out-neighbors of a_ℓ and remove it directly if it is "spanned" by the previous layers and the set $R_\ell \subseteq L_\ell$ of elements that are *not* on any augmenting path of length d_t. With this optimization, we obtain the following lemma analogous to Lemma E.7.6.

Lemma E.10.2. *Given an $S \in \mathcal{I}_{part} \cap \hat{\mathcal{I}}$ with $d_{H(S)}(s,t) = d_t$ together with a data structure $\mathcal{D}$ of Theorem E.6.1 that maintains a basis of $U \setminus S$, it takes*

$$\tilde{O}\left(\underbrace{kr}_{(i)} + \underbrace{(|S'| - |S|) \cdot d_t \sqrt{r}}_{(ii)} + \underbrace{((|S'| - |S|) \cdot d_t + r) \cdot k}_{(iii)} + \underbrace{(kr + (|S'| - |S|)) \cdot \frac{\sqrt{r}}{d_t}}_{(iv)} \right) \tag{E.3}$$

time to obtain an $S' \in \mathcal{I}_{part} \cap \hat{\mathcal{I}}$ with $d_H(S')(s,t) > d_t$, with an additional guarantee that $\mathcal{D}$ now maintains a basis of $U \setminus S'$.

We need the following observation to bound the running time of Algorithm E.6.

Observation E.10.3. *In Algorithm E.6, it holds that $B \cup R_2 \cup R_3 \cup \cdots \cup R_{d_t - 1} \in \mathcal{I}$.*

Proof of Lemma E.10.2. We analyze the running time of Algorithm E.6 first. In particular, there are four terms in Equation (E.3) which come from the following.

(i) $\tilde{O}(kr)$: It takes $\tilde{O}(kr)$ time to compute the distance layers using Lemma E.10.1 and initialize all the binary search trees $\mathcal{T}_\ell^{(i)}$'s. Computing the rank of $L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell$ also takes $\tilde{O}(kr)$ time in total since we can pre-compute query-sets of the form $L_1 \cup \cdots \cup L_k$ for each k in $\tilde{O}(kr)$ time, and each insertion to R_ℓ takes $\tilde{O}(1)$ time.

(ii) $\tilde{O}\left((|S'| - |S|) \cdot d_t \sqrt{r}\right)$: For each of the $O(|S'| - |S|)$ augmentations, it takes $\tilde{O}(r \cdot \frac{d_t}{\sqrt{r}})$ time to update the binary search trees.

(iii) $\tilde{O}(((|S'| - |S|) \cdot d_t + r) \cdot k)$: The number of elements whose out-edges are explored is bounded by $O\left((|S'| - |S|) \cdot d_t + r\right)$. This is because for each such element u, either u is included in an augmenting path of length d_t, or u is removed in Line 17. There are $O((|S'| - |S|) \cdot d_t)$ such u's in the augmenting paths. For u removed in Line 17, if $\ell = 1$, then the number of such u's is

Algorithm E.6: Blocking flow in a k-fold union exchange graph

Input: $S \subseteq U$ which partitions into $S_1, \ldots, S_k$ of independent sets and a dynamic-basis data structure $\mathcal{D}$ of $U \setminus S$

Output: $S' \in \mathcal{I}_{\text{part}} \cap \mathcal{I}_k$ with $d_{H(S')}(s,t) > d_{H(S)}(s,t)$

Guarantee: $\mathcal{D}$ maintains a basis of $U \setminus S'$ at the end of the algorithm

Build the distance layers $L_2, \ldots, L_{d_t-1}$ of $H(S)$ with Lemma E.10.1
$L_0 \leftarrow \{s\}$ and $L_{d_t} \leftarrow \{t\}$
$B \leftarrow$ the basis maintained by $\mathcal{D}$ and $L_1 \leftarrow B$
$A_\ell \leftarrow L_\ell$ for each $0 \le \ell \le d_t$
$\mathcal{T}_\ell^{(i)} \leftarrow \textsc{Initialize}(\mathcal{M}_i, S_i, Q_{S_i}, A_\ell \cap S_i)$ for each $2 \le \ell < d_t$ and $1 \le i \le k$ (Theorem E.4.1 with $\beta = \sqrt{r}/d_t$)
$D_\ell \leftarrow \emptyset$ for each $1 \le \ell < d_t$
$R_\ell \leftarrow \emptyset$ for each $2 \le \ell < d_t$
$\ell \leftarrow 0$ and $a_0 \leftarrow s$
while $\ell \ge 0$ **do**
  if $\ell < d_t$ **then**
    if $A_\ell = \emptyset$ **then break**
    if $\ell \ge 2$ *and* $\operatorname{rank}(L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell \cup \{a_\ell\}) = \operatorname{rank}(L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell)$ **then** $A_\ell \leftarrow A_\ell - a_\ell$ and **continue**
    if $\ell > 0$ **then** Find an $a_{\ell+1} := \mathcal{T}_{\ell+1}^{(i)}.\textsc{Find}(a_\ell) \neq \perp$ for some $1 \le i \le k$
    else $a_{\ell+1} \leftarrow$ an arbitrary element in A_1
    if *such an* $a_{\ell+1}$ *does not exist* **then**
      if $\ell \ge 2$ **then** $R_\ell \leftarrow R_\ell + a_\ell$ and $\mathcal{T}_\ell^{(j)}.\textsc{Delete}(a_\ell)$ where $a_\ell \in S_j$
      $A_\ell \leftarrow A_\ell - a_\ell$ and $\ell \leftarrow \ell - 1$
    else $\ell \leftarrow \ell + 1$
  else
    `// Found augmenting path` $a_1, a_2, \ldots a_\ell$
    $B \leftarrow B - a_1$, $A_1 \leftarrow A_1 - a_1$, and $D_1 \leftarrow D_1 + a_1$
    if $\mathcal{D}.\textsc{Delete}(a_1)$ *returns a replacement* x **then**
      $B_i \leftarrow B_i + x$ and $A_i \leftarrow A_i + x$
    for $i \in \{2, \ldots, d_t - 1\}$ **do**
      $D_i \leftarrow D_i + a_i$ and $A_i \leftarrow A_i - a_i$
      $\mathcal{T}_i^{(j)}.\textsc{Delete}(a_i)$ and $\mathcal{T}_i^{(j)}.\textsc{Update}(\{a_{i-1}, a_i\})$ where $a_i \in S_j$
    Augment S along $P = (s, a_1, \ldots, a_{d_t-1}, t)$
    $\ell \leftarrow 0$
return S

$O(|S'| - |S| + r)$ because there are initially $O(r)$ elements in A_1, and we add at most one to it every augmentation. If $\ell \geq 2$, then we insert it into R_ℓ, and by Line 12, the rank of $L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell$ increases after including u into R_ℓ. By Observation E.10.3, the number of such u's is bounded by $O(r)$. The term then comes from spending $O(k)$ time iterating through the k binary search trees for each of the $O((|S'| - |S|) \cdot d_t + r)$ elements whose out-neighbors are explored.

(iv) $\tilde{O}((kr + (|S'| - |S|)) \cdot \frac{\sqrt{r}}{d_t})$: The number of elements that are once in some A_ℓ is bounded by $O(kr + |S'| - |S|)$. Initially, there are $O(kr)$ elements (A_1 plus all the A_ℓ for $\ell \geq 2$), and each augmentation adds at most one element to A_1. Each of these elements is discovered by $\mathcal{T}_\ell^{(i)}.\textsc{Find}(\cdot)$ at most once, and thus we can charge the $\tilde{O}(\frac{\sqrt{r}}{d_t})$ cost to it, resulting in the fourth term of Equation (E.3).

Note that for each element whose out-neighbors are explored, any failed attempt of $\mathcal{T}_\ell^{(i)}.\textsc{Find}(\cdot)$ costs only $\tilde{O}(1)$ instead of $\tilde{O}(\frac{\sqrt{r}}{d_t})$ according to Theorem E.4.1. The $\tilde{O}(\frac{\sqrt{r}}{d_t})$ cost of a successful search is charged to term (iv) instead of (iii).

As for correctness, it suffices to show that each of the a_ℓ removed from A_ℓ because it is spanned by $L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell$ in Line 12 is not in any augmenting path of length d_t. Consider its out-neighbor $a_{\ell+1}$ with respect to the current S, and we would like to argue that $a_{\ell+1}$ is not on any augmenting path of length d_t anymore. This is because we have already explored all the out-neighbors of elements in R_ℓ. Since $a_\ell \in \text{span}(L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell)$, by Lemma E.3.4, there must exist some $u \in L_1 \cup \cdots \cup L_{\ell-1} \cup R_\ell$ with a directed edge $(u, a_{\ell+1})$. We consider two cases:

- $u \in R_\ell$. This means that we have already explored $a_{\ell+1}$, as we finished exploring all out-neighbors of u already.

- $u \in L_1 \cup \cdots \cup L_{\ell-1}$. We know that by Lemma E.3.9, both $d_{H(S)}(s, v)$ and $d_{H(S)}(v, t)$ can only increase after augmentations for all elements v. Hence $a_{\ell+1}$ cannot be part of an augmenting path of length d_t anymore, since if it was its distance to t must be $d - (\ell + 1)$, but then the distance from u to t must be at most $d - \ell$ (which is smaller than its initial distance to t at the beginning of the phase).

As a result, all of u's out-neighbors have either already been explored or do not belong to any augmenting path of length d_t. This implies that u is not on any such path either, and thus it's correct to skip and remove it from A_ℓ. This concludes the proof of Lemma E.10.2. □

Theorem E.7.2 now follows from analyzing the total running time of $O(\sqrt{\min(n, kr)})$ runs of Lemma E.10.2.

Proof of Theorem E.7.2. We initialize the dynamic-basis data structure $\mathcal{D}$ of Theorem E.6.1 on U in $\tilde{O}(n)$ time. Let $p = \min(n, kr)$ be the rank of the k-fold matroid union. Using $\mathcal{D}$, we then run $O(\sqrt{p})$ iterations of Lemma E.10.2 until $d_{H(S)}(s,t) > \sqrt{p}$. Summing the first two terms of Equation (E.3) over these $O(\sqrt{p})$ iterations gives (recall that Lemma E.3.8 guarantees that $\sum_{d=1}^{\sqrt{p}} d \cdot (|S_d| - |S_{d-1}|) = \tilde{O}(p)$)

$$\tilde{O}\left(kr\sqrt{p} + \sqrt{r} \cdot \sum_{d=1}^{\sqrt{p}} d \cdot (|S_d| - |S_{d-1}|)\right) = \tilde{O}\left(kr\sqrt{p}\right)$$

since $p\sqrt{r} \leq kr\sqrt{p}$. The third term of Equation (E.3) contributes a total running time of

$$\tilde{O}\left(\left(\sum_{d=1}^{\sqrt{p}} dk \cdot (|S_d| - |S_{d-1}|)\right) + kr\sqrt{p}\right) = \tilde{O}\left(kr\sqrt{p} + kp\right),$$

while the fourth term of Equation (E.3) sums up to

$$\tilde{O}\left(\left(\sum_{d=1}^{\sqrt{p}} kr\frac{\sqrt{r}}{d}\right) + kr\sqrt{r}\right) = \tilde{O}\left(kr\sqrt{r}\right).$$

We finish the algorithm by finding the remaining $O(\sqrt{p})$ augmenting paths one at a time with Lemma E.10.1 in a total of $\tilde{O}(kr\sqrt{p})$ time. The k-fold matroid union algorithm thus indeed runs in $\tilde{O}\left(n + kr\sqrt{\min(n, kr)} + k\min(n, kr)\right)$ time, concluding the proof of Theorem E.7.2. □

E.11 Dynamic Oracles for Specific Matroids & Applications

In this appendix, we show how to leverage known dynamic algorithms to implement the dynamic rank oracle (Definition E.1.2) efficiently for many important matroids. What we need are data structures that can maintain the rank of a set dynamically under insertions and deletions in *worst-case* update time (converting a *worst-case* data structure to *fully-persistent* can be done by the standard technique of [DSST86; Die89], paying an overhead of $O(\log n)$). Additionally, note that the data structures do not need to work against an adaptive adversary since we only ever use the *rank* of the queried sets, which is not affected by internal randomness.

In particular, for *partition, graphic, bicircular, convex transversal*, and *simple job scheduling matroids* it is possible to maintain the rank with polylog(n) update-time, and for *linear matroids* in $O(n^{1.529})$ update-time.

Together with our matroid intersection (Section E.5) and matroid union (Section E.7) algorithms, this leads to a black-box approach to solving many different problems. In fact, we can solve matroid intersection and union on any combination

of the above matroids, leading to improved or matching running times for many problems (see the introduction Section E.1 with Section E.1 and Table E.2 for a more thorough discussion). For completeness, we define these problems in Section E.11.5. The same algorithms are powerful enough to also solve new problems which have not been studied before.

Example Application: Tree Scheduling (or Maximum Forest with Deadlines). We give an example of a reasonably natural combinatorial scheduling problem, which—to our knowledge—has not been studied before. Suppose we are given a graph $G = (V, E)$ where each edge $e \in E$ has two numbers associated with it: a release date ℓ_e and a deadline r_e. Consider the problem where we want to for each day pick exactly one edge (say, to build/construct), but we have constraints that edge e can only be picked between days ℓ_e and r_e. Now the task is to come up with a scheduling plan to build a spanning tree of the graph, if possible.

This problem is exactly a matroid intersection problem between a graphic matroid and a convex transversal matroid. Hence, by a black-box reduction, we know that we can solve this problem in $\tilde{O}(|E|\sqrt{|V|})$ time.

E.11.1 Partition Matroids

In a partition matroid $\mathcal{M} = (U, \mathcal{I})$, each element $u \in U$ is assigned a color c_u. We are also, for each color c, given a non-negative integer d_c, and we define a set of elements $S \subseteq U$ to be independent if for each color c, S includes at most d_c elements of this color. Implementing the dynamic oracle for the partition matroid is easy:

Lemma E.11.1. *One can maintain the rank of a partition matroid in $O(1)$-update time.*

Proof. For each color c we maintain a counter x_c of how many elements we have of color c. We also maintain $r = \sum \min_c(x_c, d_c)$, which is the rank of the current set. □

Remark E.11.2. Bipartite matching can be modeled as a matroid intersection problem of two partition matroids. So our matroid intersection algorithm together with the above lemma match (up to poly-logarithmic factors induced by fully-persistence)—in a black-box fashion—the $O(|E|\sqrt{|V|})$-time bound of the best *combinatorial* algorithm for bipartite matching [HK73].

E.11.2 Graphic and Bicircular Matroids

Given a graph $G = (V, E)$, the *graphic* and *bicircular* matroids are matroids capturing the connectivity structure of the graph.

Graphic Matroid. In the graphic matroids $\mathcal{M} = (E, \mathcal{I})$ a subset of edges $E' \subseteq E$ are independent if and only if they do not contain a cycle. We use the following result to implement the dynamic oracle for this matroid.

Lemma E.11.3 ([KKM13; GKKT15])**.** *There is a data structure that maintains an initially-empty graph $G = (V, E)$ and supports insertion/deletion of edges e into/from E in worst case $O(\log^4 |V|)$ time and query of the connectivity between u and v in worst case $O(\log |V| / \log \log |V|)$ time. The data structure works with high probability against an oblivious adversary.*

With a simple and standard extension, we can maintain the number of connected components as well, and hence also the rank (since $\text{rank}(E') = |V| - \#\text{connected components in } G[E']$).

Corollary E.11.4. *There is a data structure that maintains an initially-empty graph $G = (V, E)$ and supports insertion/deletion of e into/from E in worst-case $O(\log^4 |V|)$ time. After each operation, the data structure also returns the number of connected components in G. The data structure works with high probability against an oblivious adversary.*

Proof. We maintain the data structure $\mathcal{C}$ of Lemma E.11.3 and a counter $c := |V|$ representing the number of connected components. For insertion of $e = (u, v)$, we first query the connectivity of u and v before inserting e into $\mathcal{C}$. If they are not connected before the insertion, decrease c by one. For deletion of $e = (u, v)$, after deleting e from $\mathcal{C}$, we check if u and v are still connected. If not, then we increase c by one. □

Bicircular Matroid. In the bicircular matroid $\mathcal{M} = (E, \mathcal{I})$, a subset of edges $E' \subseteq E$ are independent if and only if each connected component in $G[E']$ has at most one cycle. Similar to the graphic matroid, dynamic connectivity algorithms can be used to implement the dynamic rank oracle for bicircular matroids too.

Corollary E.11.5. *There is a data structure that maintains an initially-empty graph $G = (V, E)$ and supports insertion/deletion of e into/from E in worst-case $O(\log^4 |V|)$ time. After each operation, the data structure also returns the rank of E in the bicircular matroid. The data structure works with high probability against an oblivious adversary.*

Proof. The dynamic connectivity data structure (Lemma E.11.3) of [KKM13; GKKT15] can be adapted to also keep track of the number of edges and vertices in each connected component. Using this, the data structure can, for each connected component c keep track of a number x_c as the minimum of the number of edges in this component and the number of vertices in this component. Then the rank of the bicircular matroid is just the sum of x_c (as in an independent set each component is either a tree or a tree with an extra edge). In each update two

components can merge, a component can be split up into two, or the edge-count of a component may simply change. □

Remark E.11.6 (Deterministic Dynamic Connectivity). The above dynamic connectivity data structures are randomized. There are also deterministic connectivity data structures, but with slightly less efficient sub-polynomial $|V|^{o(1)}$ update time [CGLNPS20].

E.11.3 Convex Transversal and Scheduling Matroids

Convex transversal and scheduling matroids are special cases of the *transversal* matroid, with applications in scheduling algorithms.

Definition E.11.7 (Transversal Matroid [EF65])**.** A *transversal matroid* with respect to a bipartite graph $G = (L, R, E)$ is defined over the ground set L, where each $S \subseteq L$ is independent if and only if there is a perfect matching in G between S and a subset of R.

A bipartite graph $G = (L, R, E)$ is *convex* if R has a linear order $R = \{r_1, r_2, \ldots, r_n\}$ and each $\ell \in L$ corresponds to an interval $1 \leq s(\ell) \leq t(\ell) \leq n$ such that $(\ell, r_i) \in E$ if and only if $s(\ell) \leq i \leq t(\ell)$, i.e., the neighbors of each ℓ form an interval.

Definition E.11.8 (Convex Transversal Matroid and Simple Job Scheduling Matroid)**.** A *convex transversal matroid* is a transversal matroid with respect to a convex bipartite graph. A *simple job scheduling matroid* is a special case of convex transversal matroids in which $s(\ell) = 1$ for each $\ell \in L$.

One intuitive way to think about the simple job scheduling matroid is that there is a machine capable of finishing one job per day. The ground set of the matroid consists of n jobs, where the i-th jobs must be done before its deadline d_i. A subset of jobs forms an independent set if it's possible to schedule these jobs on the machine so that every job is finished before its deadline.

Lemma E.11.9 (Dynamic Convex Bipartite Matching [BGHK07])**.** *There is a data structure which, given a convex bipartite graph $G = (L, R, E)$, after $\tilde{O}(|L| + |R|)$ initialization, maintains the size of the maximum matching of $G[A \cup R]$ where $A \subseteq L$ is a dynamically changing subset of L that is initially empty. The data structure supports insertion/deletion of an $x \in L$ to/from A in worst-case $O(\log^2(|L| + |R|))$ update time.*

Remark E.11.10. The exact data structure presented in [BGHK07] is different from the stated one. In particular, they support insertion/deletion of an *unknown* job, i.e., we do not know beforehand what the starting date and deadline of the job are, nor do we know its relative position among the current set of jobs. As a result, they used a rebalancing-based or rebuilding-based binary search tree [NR72; And89;

And91], resulting in their amortized bound. For our use case, all the possible jobs are known and we are just activating/deactivating them, hence a static binary tree with a worst-case guarantee over these jobs suffices.

E.11.4 Linear Matroid

In a linear matroid $\mathcal{M} = (U, \mathcal{I})$, U is a set of n vectors (of dimension r) in some vector space and the notion of independence is just that of *linear independence*. The dynamic algorithm to maintain the rank of a matrix of [BNS19] can be used without modification as the dynamic oracle.

Lemma E.11.11 (Dynamic Matrix Rank Maintenance [BNS19])**.** *There is a data structure which, given an $n \times n$ matrix M, maintains the rank of M under row updates in worst-case $O(n^{1.529})$ update time.*

E.11.5 Problems

For completeness, here we define the problems we discuss in the introduction, and why they reduce to matroid union or intersection.

k-Forest. In this problem we are given a graph $G = (V, E)$ and asked to find k edge-disjoint forests of the graph, of the maximum total size. It can be modeled as the k-fold matroids union over the graphic matroid of G.

k-Disjoint Spanning Trees. This problem is a special case of the above k-forest problem where we ask to find k edge-disjoint spanning trees of the graph. Clearly, if such exists, the k-forest problem will find them.

k-Pseudoforest. Similar to above, in this problem we are given a graph $G = (V, E)$ and asked to find k edge-disjoint *pseudoforests* of the graph, of the maximum total size. A pseudoforest is an undirected graph in which every component has at most one cycle. The problem can be modeled as the k-fold matroids union over the bicircular matroid of G.

(f, p)-Mixed Forest-Pseudoforest. Again, we are given a graph $G = (V, E)$ and asked to find f forests and p pseudoforest (all edge-disjoint), of the maximum total size. The problem can be modeled as the matroids union over f graphic matroids and p bicircular matroids.

Tree Packing. In the tree packing problem, we are given a graph $G = (V, E)$ and are asked to find the maximum k such that we can find k-disjoint spanning trees in the graph. This number k is sometimes called the *tree-pack-number* or *strength* of the graph. The problem can be solved with the k-disjoint spanning trees problem,

by binary searching for k in the range $[0, |E|/(|V|-1)]$, and is an example of a *matroid packing* problem.

Arboricity and Pseudoarboricity. The arboricity (respectively pseudoarboricity) of a graph $G = (V, E)$ is the least integer k such that we can partition the edges into k edge-disjoint forests (respectively pseudoforests). This can be solved with the k-forest (respectively k-pseudoforest) problem with a binary search over k. It is well known that for a simple graph the (pseudo-)arboricity is at most $\sqrt{|E|}$, so we need only search for k in the range $[0, \sqrt{|E|}]$. The problems are examples of *matroid covering* problems.

Shannon Switching Game. The Shannon switching game is a game played on a graph $G = (V, E)$, between two players "Short" and "Cut". They alternate turns with Short playing first, and all edges are initially colored white. On Short's turn, he may color an edge of the graph black. On Cut's turn, he picks a remaining non-black edge and removes it from the graph. Short wins if he connects the full graph with black edges, and Cut if he manages to disconnect the graph. It can be shown that Short wins if and only if there exists two disjoint spanning trees in the graph (and these two spanning trees describes a winning strategy for Short). Hence solving this game is a special case of the k-disjoint spanning tree problem with $k = 2$.

Graph k-Irreducibility. A (multi-)graph $G = (V, E)$ is called k-irreducible ([Whi88]) if and only if $|E| = k(|V| - 1)$ and for any vertex-induced nonempty, proper subgraph $G[V']$ it holds that $|E(G[V'])| < k(|V'|-1)$. The motivation behind this definition comes from the rigidity of *bar-and-body frameworks*. A bar-and-body framework where rigid bars are attached to rigid bodied with joints (represented by the graph G). Then any stress put on a k-irreducible structure will propagate to all the bars (i.e. edges). [GW88] show how one can decide if a graph is k-irreducible by first determining if its edges can be partitioned into k edge-disjoint trees, and then performing an additional $\tilde{O}(k|V|)$ work.

Bipartite Matching. In the bipartite matching problem, we are given a bipartite graph $G = (L \cup R, E)$, and the goal is to find a *matching* (a set of edges which share no vertices) of maximum size. Bipartite matching can be modeled as a matroid intersection problem over two partition matroids $M_L = (E, \mathcal{I}_L)$ and $M_R = (E, \mathcal{I}_R)$. M_L specifies that no two edges share the same vertex on the left L (and M_R is defined similarly on the right set of vertices R).

Colorful Spanning Tree. In this problem[27], we are given a graph $G = (V, E)$ together with colors on the edges $c : E \to \mathbb{Z}$. We are tasked to find a spanning

[27] sometimes also called *rainbow spanning tree*.

tree of G such that no two edges in our spanning tree have the same color. This problem can be modeled by the matroid intersection of the graphic matroid of G (ensuring we pick a forest), and a partition matroid of the coloring c (ensuring that we pick no duplicate colors). We also note that this problem is more difficult than bipartite matching since any bipartite matching instance can be converted to a colorful spanning tree instance on a star-multi-graph.

Graphic Matroid Intersection. In graphic matroid intersection we are given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and a bijection of the edges $\phi : E_1 \to E_2$. The task is to find a forest in G_1 of the maximum size, which also maps to a forest in G_2. By definition, this is a matroid intersection problem over two graphic matroids. Again, this problem is a further generalization of the colorful spanning tree problem.

Convex Transversal and Simple Job Scheduling Matroid Intersection. In these problems, we are given a set of unit-size jobs V, where each job v has two release times $\ell_1(v)$, $\ell_2(v) \geq 1$ (in simple job scheduling $\ell(v) = 1$) and two deadlines $r_1(v)$,$r_2(v) \leq \mu$. The task is to find a set of jobs S of the maximum size such that they can be scheduled on two machines as follows: each job needs to be scheduled at both machines, and at machine i it must be scheduled at time $t \in [\ell_i(v), r_i(v)]$.

Linear Matroid Intersection . In this problem, we are given two $n \times r$ matrices M_1 and M_2 over some field. The task is to find a set of indices $S \subseteq \{1, 2, \ldots, n\}$ of maximum cardinality, such that the rows of M_1 (respectively M_2) indexed by S are independent at the same time. This is a matroid intersection of two linear matroids defined by M_1 and M_2. We note that partition, graphic, and transversal matroids are special cases of linear matroids.

E.12 Independence-Query Matroid Intersection Algorithm

In this section, we show that we can obtain an $\tilde{O}(nr^{3/4})$ matroid intersection algorithm in the dynamic-independence-oracle model. This matches the state-of-the-art traditional independence-query algorithm of Blikstad [Bli21]. We will only provide a proof sketch here because our algorithm is mostly an implementation of [Bli21] in the new model with the help of (circuit) binary search trees.

Using the same construction as Theorem E.4.1 and Observation E.4.3, circuit binary search trees work verbatim in the dynamic-independence-oracle model (however, co-circuit binary search trees do not). In particular, Observation E.4.3(iii) can be checked with a single independence query.

Corollary E.12.1. *For any integer $\beta \geq 1$, there exists a data structure that supports the following operations.*

- *INITIALIZE*$(\mathcal{M}, S, Q_S, X)$*: Given* $S \in \mathcal{I}$*, the query-set* Q_S *that corresponds to* S*, and* $X \subseteq S$ *or* $X = \{t\}$*, initialize the data structure in* $\tilde{O}(|X|)$ *time. The data structure also maintains* S*.*

- *FIND*(y)*: Given* $y \in \bar{S}$*,*

  - *if* $X \subseteq S$*, then return an* $x \in X$ *such that* $S - y + x \in \mathcal{I}$*, or*
  - *if* $X = \{t\}$*, then return the only element* $x = s$ *or* $x = t$ *in* X *if* $S + y \in \mathcal{I}$ *and* $\perp$ *otherwise.*

  The procedure returns $\perp$ *if such an* x *does not exist. The procedure takes* $\tilde{O}(\beta)$ *time if the result is not* $\perp$*, and* $\tilde{O}(1)$ *time otherwise.*

- *DELETE*(x)*: Given* $x \in X$*, if* $x \notin \{s, t\}$*, delete* x *from* X *in* $O(\log n)$ *time.*

- *REPLACE*(x, y)*: Given* $x \in X$ *and* $y \notin X$*, replace* x *in* X *by* y *in* $O(\log n)$ *time.*

- *UPDATE*(Δ)*: Update* S *to* $S \oplus (\Delta \setminus \{s, t\})$ *in amortized* $\tilde{O}(\frac{|X| \cdot |\Delta|}{\beta})$ *time.*

Framework. The algorithm of [Bli21] consists of the following three phases.

1. First, obtain an $(1 - \epsilon)$-approximate solution S using augmenting sets in $\tilde{O}(\frac{n\sqrt{r}}{\epsilon})$ time.

2. Eliminate all augmenting paths in $G(S)$ of length at most d using Cunningham's algorithm implemented by [CLSSW19] in $\tilde{O}(nd + nr\epsilon)$ time.

3. Finding the remaining $O(r/d)$ augmenting paths one at a time, using $\tilde{O}(n\sqrt{r})$ time each.

With $\epsilon = r^{-1/4}$ and $d = r^{3/4}$, the total running time is $\tilde{O}(nr^{3/4})$. We briefly sketch how to implement the above three steps in the same running time also for the dynamic-independence-oracle model.

Note that the primary difficulty independence-query algorithms face is that we are only capable of checking Observation E.4.3(iii) (using Corollary E.12.1), which means that we can only explore the neighbors of $u \in \bar{S}$. The aforementioned rank-query algorithms for building distance layers and finding blocking-flow style augmenting paths are thus inapplicable in the dynamic-independence-oracle model.

Approximation Algorithm. The $O(n\sqrt{r}/\epsilon)$-query $(1-\epsilon)$-approximation algorithm of [Bli21] needs to first compute distance layers up to distance $O(\frac{1}{\epsilon})$. This is done in a similar way as sketched below for "Eliminating Short Augmenting Paths".

Otherwise, the approximation algorithm works through a series of "refine" operations (algorithms `RefineAB`, `RefineBA`, and `RefineABA` in [CLSSW19; Bli21]) to build a partial augmenting set. In each such operation, we only need to be able to do the following for some sets (P, Q): start from some set Q and find a maximal set $X \subseteq P$ such that $Q + X$ is independent. This can be performed with a greedy algorithm in time (and dynamic query) $O(|P|)$, given that we already have queried set Q before (which will be the case).

Finally, the approximation algorithm falls back to finding a special type of augmenting paths *with respect to* the current augmenting set, in the `RefinePath` algorithm of [Bli21], with $\tilde{O}(n)$ queries for each such path. This algorithm can be implemented also in the dynamic-oracle model with the same query complexity. `RefinePath` relies on the `RefineAB` and `RefineBA` algorithms (which we already covered), in addition to a binary search trick to find feasible exchange pairs. This binary search can be implemented with the circuit trees (Corollary E.12.1), and it takes a total time $\tilde{O}(n)$ to build them (since we can keep track of a queried set for S, and then we only need to build a circuit tree statically once for each layer in cost proportional to the size of the layer—which sums up to $\tilde{O}(n)$).

Eliminating Short Augmenting Paths. Using [CLSSW19]'s implementation of Cunningham's algorithm, we can eliminate all augmenting paths of length d, thereby obtaining a $(1-1/d)$-approximate solution. The algorithm relies on Lemma E.3.9 to "fix" the distance layers after each augmentation. Initially, all elements have distance 1 or 2 from s depending on whether it belongs to S (the common independent set obtained by the above approximation algorithm) or not. Before the first and after each of the remaining $O(\epsilon r)$ augmentation, we can fix the distance layers as follows.

- For each $1 \le \ell \le d$ and $u \in L_\ell$, if u is not of distance ℓ from s, i.e., there is no in-edge from $L_{\ell-1}$ to u anymore, move u from L_ℓ to $L_{\ell+2}$. This check is done as follows, depending on the parity of ℓ.
  - If ℓ is even, then for each $v \in L_{\ell-1}$, we find all the unmarked $u \in L_\ell$ that v has an edge to and mark u. In the end, all the unmarked $u \in L_\ell$ do not belong to L_ℓ and should be moved to $L_{\ell+2}$.
  - If ℓ is odd, then we simply check if there is an in-edge from $L_{\ell-1}$ to decide whether u should be moved to other layers.

Both cases can be implemented efficiently with circuit binary search trees of Corollary E.12.1: Each time we spend $\tilde{O}(1)$ time to either confirm that u has distance ℓ from s with respect to the current S (in which case it will not be moved anymore in this iteration), or we increase the distance estimate of u. The total running time is thus $\tilde{O}(nd + nr\epsilon)$, where $\tilde{O}(nd)$ comes from increasing the distance

estimate of each element to at most d, and $\tilde{O}(nr\epsilon)$ comes from confirming that each element belongs to its distance layer in the $O(\epsilon r)$ iterations.

A caveat here is that we need to support insertion/deletion into the binary search trees. This can be made efficient by doubling the size of a binary search tree (and re-initializing it) every time when there are not enough leaf nodes left in it. The cost of re-building will be amortized to $\tilde{O}(1)$ time per update (i.e., movement of an element to another layer).

Finding a Single Augmenting Path. With the $(1 - 1/d)$-approximate solution obtained in the first two steps, [Bli21] then finds the remaining $O(r/d)$ augmenting paths one at a time, using the *reachability* algorithm of [BBMN21]. The reachability algorithm roughly goes as follows. First, we initialize two circuit binary search trees (Corollary E.12.1) over the two matroids for discovering out-edges and in-edges of elements in $\bar{S}$. We then repeatedly run the following three steps until either an (s, t)-path is found (an arbitrary (s, t)-path suffices since such a path can be converted into a chordless one in $\tilde{O}(r)$ time along which augmentation is valid) or we conclude that t is unreachable from s. We keep track of a set of visited vertices F which we know are reachable from s.

(i) Identify the set of unvisited *heavy* vertices in $\bar{S}$ that have at least $\sqrt{r}$ unvisited out-neighbors or has a direct edge toward t. This is done by sampling a set R of unvisited vertices in S and then computing for each vertex u whether $R \cap \mathsf{OutNgh}(u) = \emptyset$, or equivalently, whether $S - R + u \in \mathcal{I}$. Intuitively, vertices with more out-neighbors are more likely to fail the test. This can be tested for a single R and all u in $O(n)$ time in the dynamic-independence-oracle model. With $O(\log n)$ samples, heavy vertices can be successfully identified with high probability.

(ii) Discover all the out-neighbors for each light vertex, taking a total of $\tilde{O}(n\sqrt{r})$ time using the circuit binary search tree over the whole run of the algorithm. (Each vertex turns from heavy to light at most once.)

(iii) Perform a reversed breadth-first-search from all the heavy vertices simultaneously. We can assume that every vertex on the path is light (i.e., we find a "closest" heavy vertex reachable from s), and thus all its out-neighbors have already been discovered. That is, going backward from S to $\bar{S}$, we use the out-edges of light vertices. From $\bar{S}$ to S, we use the circuit binary search tree. This takes $\tilde{O}(n)$ time, and we either find a heavy vertex reachable from s (in which case we make progress by visiting at least $\sqrt{r}$ vertices in S), or we conclude that all heavy vertices are unreachable from s (in which case t is unreachable either).

The number of iterations is bounded by $O(\sqrt{r})$ since we discover at least $\sqrt{r}$ unvisited vertices in S every time. The total running time of finding a single augmenting path is thus $\tilde{O}(n\sqrt{r})$.

Using the same parameters ϵ and d as in [Bli21] to combine the three phases, we obtain the following matroid intersection algorithm in the dynamic-independence-oracle model.

Theorem E.12.2. *For two matroids $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$, it takes $\tilde{O}(nr^{3/4})$ time to obtain the largest $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with high probability in the dynamic-independence-oracle model.*

Bibliography

[AB21] Daniel Anderson and Guy E. Blelloch. "Parallel Minimum Cuts in $O(m \log^2 n)$ Work and Low Depth". In: *SPAA*. ACM, 2021, pp. 71–82. DOI: 10.1145/3409964.3461797 (cit. on p. 309).

[AD71] Martin Aigner and Thomas A. Dowling. "Matching Theory for Combinatorial Geometries". In: *Transactions of the American Mathematical Society* 158.1 (1971), pp. 231–245. DOI: 10.2307/1995784 (cit. on p. 276).

[AEGLMN22] Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Cut Query Algorithms with Star Contraction". In: *FOCS*. IEEE, 2022, pp. 507–518. DOI: 10.1109/FOCS54457.2022.00055 (cit. on pp. 275, 309).

[AKLPST22] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. "Breaking the Cubic Barrier for All-Pairs Max-Flow: Gomory-Hu Tree in Nearly Quadratic Time". In: *FOCS*. IEEE, 2022, pp. 884–895. DOI: 10.1109/FOCS54457.2022.00088 (cit. on p. 269).

[AKT21a] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. "APMF ¡ APSP? Gomory-Hu Tree for Unweighted Graphs in Almost-Quadratic Time". In: *FOCS*. IEEE, 2021, pp. 1135–1146. DOI: 10.1109/FOCS52979.2021.00112 (cit. on p. 269).

[AKT21b] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. "Subcubic algorithms for Gomory-Hu tree in unweighted graphs". In: *STOC*. ACM, 2021, pp. 1725–1737. DOI: 10.1145/3406325.3451073 (cit. on p. 269).

[AKT22] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. "Friendly Cut Sparsifiers and Faster Gomory-Hu Trees". In: *SODA*. SIAM, 2022, pp. 3630–3649. DOI: 10.1137/1.9781611977073.143 (cit. on p. 269).

[AMV20] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. "Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs". In: *FOCS*. IEEE, 2020, pp. 93–104. DOI: 10.1109/FOCS46700.2020.00018 (cit. on p. 275).

[AMV21] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. "Faster Sparse Minimum Cost Flow by Electrical Flow Localization". In: *FOCS*. IEEE, 2021, pp. 528–539. DOI: 10.1109/FOCS52979.2021.00059 (cit. on p. 269).

[And89] Arne Andersson. "Improving Partial Rebuilding by Using Simple Balance Criteria". In: *WADS*. Vol. 382. Lecture Notes in Computer Science. Springer, 1989, pp. 393–402. DOI: 10.1007/3-540-51542-9_33 (cit. on pp. 289, 317).

[And91] Arne Andersson. "Maintaining α-balanced trees by partial rebuilding". In: *Int. J. Comput. Math.* 38.1-2 (1991), pp. 37–48. DOI: 10.1080/00207169108803956 (cit. on pp. 289, 318).

[AW21] Josh Alman and Virginia Vassilevska Williams. "A Refined Laser Method and Faster Matrix Multiplication". In: *SODA*. SIAM, 2021, pp. 522–539. DOI: 10.1137/1.9781611976465.32 (cit. on p. 273).

[BBEMN22] Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Nearly Optimal Communication and Query Complexity of Bipartite Matching". In: *FOCS*. IEEE, 2022, pp. 1174–1185. DOI: 10.1109/FOCS54457.2022.00113 (cit. on p. 308).

[BBMN21] Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Breaking the quadratic barrier for matroid intersection". In: *STOC*. ACM, 2021, pp. 421–432. DOI: 10.1145/3406325.3451092 (cit. on pp. 271, 284, 323).

[BF20] Markus Blumenstock and Frank Fischer. "A Constructive Arboricity Approximation Scheme". In: *SOFSEM*. Vol. 12011. Lecture Notes in Computer Science. Springer, 2020, pp. 51–63. DOI: 10.1007/978-3-030-38919-2_5 (cit. on p. 269).

[BGHK07] Gerth Stølting Brodal, Loukas Georgiadis, Kristoffer Arnsfelt Hansen, and Irit Katriel. "Dynamic Matchings in Convex Bipartite Graphs". In: *MFCS*. Vol. 4708. Lecture Notes in Computer Science. Springer, 2007, pp. 406–417. DOI: 10.1007/978-3-662-48054-0_50 (cit. on p. 317).

[BGJLLPS22] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. "Faster maxflow via improved dynamic spectral vertex sparsifiers". In: *STOC*. ACM, 2022, pp. 543–556. DOI: 10.1145/3519935.3520068 (cit. on p. 269).

[BHKP08] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. "Fast edge splitting and Edmonds' arborescence construction for unweighted graphs". In: *SODA*. SIAM, 2008, pp. 455–464 (cit. on p. 308).

[Bli21] Joakim Blikstad. "Breaking O(nr) for Matroid Intersection". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 31:1–31:17. DOI: 10.4230/LIPIcs.ICALP.2021.31 (cit. on pp. 271, 273, 276, 320–324).

[BLLSSSW21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances". In: *STOC*. ACM, 2021, pp. 859–869. DOI: 10.1145/3406325.3451108 (cit. on pp. 269, 275).

[BLNPSSSW20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs". In: *FOCS*. IEEE, 2020, pp. 919–930. DOI: 10.1109/FOCS46700.2020.00090 (cit. on pp. 269, 275, 308).

[BNS19] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. "Dynamic Matrix Inverse: Improved Algorithms and Matching Conditional Lower Bounds". In: *FOCS*. IEEE Computer Society, 2019, pp. 456–480. DOI: 10.1109/FOCS.2019.00036 (cit. on p. 318).

[CGJS22] Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. "Improved Lower Bounds for Submodular Function Minimization". In: *FOCS*. IEEE, 2022, pp. 245–254. DOI: 10.1109/FOCS54457.2022.00030 (cit. on p. 309).

[CGLNPS20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. "A Deterministic Algorithm for Balanced Cut with Applications to Dynamic Connectivity, Flows, and Beyond". In: *FOCS*. IEEE, 2020, pp. 1158–1167. DOI: 10.1109/FOCS46700.2020.00111 (cit. on pp. 272, 317).

[CHLP23] Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. "Steiner Connectivity Augmentation and Splitting-off in Poly-logarithmic Maximum Flows". In: *SODA*. SIAM, 2023, pp. 2449–2488. DOI: 10.1137/1.9781611977554.ch95 (cit. on p. 269).

[CKLPGS22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *FOCS*. IEEE, 2022, pp. 612–623. DOI: 10.1109/FOCS54457.2022.00064 (cit. on pp. 269, 274, 275, 308, 309).

[CLNPSQ21] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. "Minimum Cuts in Directed Graphs via Partial Sparsification". In: *FOCS*. IEEE, 2021, pp. 1147–1158. DOI: 10.1109/FOCS52979.2021.00113 (cit. on pp. 269, 309).

[CLP22] Ruoxu Cen, Jason Li, and Debmalya Panigrahi. "Augmenting Edge Connectivity via Isolating Cuts". In: *SODA*. SIAM, 2022, pp. 3237–3252. DOI: 10.1137/1.9781611977073.127 (cit. on p. 269).

[CLSSW19] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. "Faster Matroid Intersection". In: *FOCS*. IEEE Computer Society, 2019, pp. 1146–1168. DOI: 10.1109/FOCS.2019.00072 (cit. on pp. 271, 273, 275, 276, 278–280, 284, 285, 287, 290–293, 304, 321, 322).

[CMSV17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. "Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}(m^{10/7} \log W)$ Time (Extended Abstract)". In: *SODA*. SIAM, 2017, pp. 752–771. DOI: 10.1137/1.9781611974782.48 (cit. on p. 275).

[Cun86] William H. Cunningham. "Improved Bounds for Matroid Partition and Intersection Algorithms". In: *SIAM J. Comput.* 15.4 (1986), pp. 948–957. DOI: 10.1137/0215066 (cit. on pp. 273, 276, 278–280, 284, 299).

[Die89] Paul F. Dietz. "Fully Persistent Arrays (Extended Array)". In: *WADS*. Vol. 382. Lecture Notes in Computer Science. Springer, 1989, pp. 67–74. DOI: 10.1007/3-540-51542-9_8 (cit. on pp. 272, 314).

[Din70] Efim A Dinic. "Algorithm for solution of a problem of maximum flow in networks with power estimation". In: *Soviet Math. Doklady*. Vol. 11. 1970, pp. 1277–1280 (cit. on pp. 276, 278).

[DSST86] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. "Making Data Structures Persistent". In: *STOC*. ACM, 1986, pp. 109–121. DOI: 10.1145/12130.12142 (cit. on pp. 272, 314).

[Edm70] Jack Edmonds. "Submodular functions, matroids, and certain polyhedra". In: *Combinatorial structures and their applications*. Gordon and Breach, 1970, pp. 69–87 (cit. on p. 276).

[Edm71] Jack Edmonds. "Matroids and the greedy algorithm". In: *Math. Program.* 1.1 (1971), pp. 127–136 (cit. on p. 294).

[EDVJ68] Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. "Matroid partition". In: *50 Years of Integer Programming 1958–2008* (1968), p. 199 (cit. on pp. 280, 299).

[EF65] Jack Edmonds and Delbert Ray Fulkerson. "Transversals and matroid partition". In: *J. Res. Nat. Bur. Standards Sect. B* 69 (1965), pp. 147–153. DOI: 10.6028/jres.069b.016 (cit. on pp. 274, 317).

[EGIN97] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. "Sparsification - a technique for speeding up dynamic graph algorithms". In: *J. ACM* 44.5 (1997), pp. 669–696. DOI: 10.1145/265910.265914 (cit. on pp. 277, 281, 282, 296).

[Fre85] Greg N. Frederickson. "Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications". In: *SIAM J. Comput.* 14.4 (1985), pp. 781–798. DOI: 10.1137/0214055 (cit. on pp. 277, 281, 296).

[FS89] Greg N. Frederickson and Mandayam A. Srinivas. "Algorithms and Data Structures for an Expanded Family of Matroid Intersection Problems". In: *SIAM J. Comput.* 18.1 (1989), pp. 112–138. DOI: 10.1137/0218008 (cit. on p. 271).

[Gab91] Harold N. Gabow. "A Matroid Approach to Finding Edge Connectivity and Packing Arborescences". In: *STOC*. ACM, 1991, pp. 112–122. DOI: 10.1145/103418.103436 (cit. on pp. 269, 271, 273).

[Gab95] Harold N. Gabow. "A Matroid Approach to Finding Edge Connectivity and Packing Arborescences". In: *J. Comput. Syst. Sci.* 50.2 (1995), pp. 259–273. DOI: 10.1006/jcss.1995.1022 (cit. on pp. 269, 273, 274).

[Gar61] Martin Gardner. *The Second Scientific American Book of Mathematical Puzzles and Diversions*. 1961 (cit. on p. 269).

[GKKT15] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. "Dynamic graph connectivity with improved worst case update time and sublinear space". In: *CoRR* abs/1509.06464 (2015). arXiv: 1509.06464 (cit. on pp. 272, 304, 316).

[GLP21] Yu Gao, Yang P. Liu, and Richard Peng. "Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao". In: *FOCS*. IEEE, 2021, pp. 516–527. DOI: 10.1109/FOCS52979.2021.00058 (cit. on p. 269).

[GS85] Harold N. Gabow and Matthias F. M. Stallmann. "Efficient Algorithms for Graphic Matroid Intersection and Parity (Extended Abstract)". In: *ICALP*. Vol. 194. Lecture Notes in Computer Science. Springer, 1985, pp. 210–220. DOI: 10.1007/BFb0015746 (cit. on pp. 269, 271, 274, 278).

[GSS93] Jack E Graver, Brigitte Servatius, and Herman Servatius. *Combinatorial rigidity*. 2. American Mathematical Soc., 1993 (cit. on p. 269).

[GT79] Harold N. Gabow and Robert Endre Tarjan. "Efficient Algorithms for Simple Matroid Intersection Problems". In: *FOCS*. IEEE Computer Society, 1979, pp. 196–204. DOI: 10.1109/SFCS.1979.14 (cit. on p. 271).

[GW88] Harold Gabow and Herbert Westermann. "Forests, frames, and games: algorithms for matroid sums and applications". In: *STOC*. 1988, pp. 407–421. DOI: 10.1145/62212.62252 (cit. on pp. 269, 271, 273, 274, 281, 304, 319).

[GX89] Harold N. Gabow and Ying Xu. "Efficient Algorithms for Independent Assignments on Graphic and Linear Matroids". In: *FOCS*. IEEE Computer Society, 1989, pp. 106–111. DOI: 10.1109/SFCS.1989.63463 (cit. on pp. 269, 271, 274).

[Har08] Nicholas J. A. Harvey. "Matroid intersection, pointer chasing, and Young's seminormal representation of S_n". In: *SODA*. SIAM, 2008, pp. 542–549 (cit. on pp. 275, 304, 305).

[Har09] Nicholas J. A. Harvey. "Algebraic Algorithms for Matching and Matroid Problems". In: *SIAM J. Comput.* 39.2 (2009), pp. 679–702. DOI: 10.1137/070684008 (cit. on p. 274).

[HK73] 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. DOI: 10.1137/0202019 (cit. on pp. 274–276, 278, 290, 291, 308, 315).

[HMT88] András Hajnal, Wolfgang Maass, and György Turán. "On the Communication Complexity of Graph Properties". In: *STOC*. ACM, 1988, pp. 186–191. DOI: 10.1145/62212.62228 (cit. on pp. 277, 306, 307).

[HSV21] David G. Harris, Hsin-Hao Su, and Hoa T. Vu. "On the Locality of Nash-Williams Forest Decomposition and Star-Forest Decomposition". In: *PODC*. ACM, 2021, pp. 295–305. DOI: 10.1145/3465084.3467908 (cit. on p. 269).

[JLSW20] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. "An improved cutting plane method for convex optimization, convex-concave games, and its applications". In: *STOC*. ACM, 2020, pp. 944–953. DOI: 10.1145/3357713.3384284 (cit. on p. 309).

[Kar98] David R. Karger. "Random sampling and greedy sparsification for matroid optimization problems". In: *Math. Program.* 82 (1998), pp. 41–81. DOI: 10.1007/BF01585865 (cit. on p. 269).

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. "Dynamic graph connectivity in polylogarithmic worst case time". In: *SODA*. SIAM, 2013, pp. 1131–1142. DOI: 10.1137/1.9781611973105.81 (cit. on pp. 272, 304, 316).

[KLS20] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. "Unit Capacity Maxflow in Almost $O(m^{4/3})$ Time". In: *FOCS*. IEEE, 2020, pp. 119–130. DOI: 10.1109/FOCS46700.2020.00020 (cit. on p. 269).

[Law75] Eugene L. Lawler. "Matroid intersection algorithms". In: *Math. Program.* 9.1 (1975), pp. 31–56. DOI: 10.1007/BF01681329 (cit. on p. 276).

[LLSZ21] Troy Lee, Tongyang Li, Miklos Santha, and Shengyu Zhang. "On the Cut Dimension of a Graph". In: *CCC*. Vol. 200. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 15:1–15:35. DOI: 10.4230/LIPIcs.CCC.2021.15 (cit. on p. 309).

[LMN21] Andrés López-Martínez, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Work-Optimal Parallel Minimum Cuts for Non-Sparse Graphs". In: *SPAA*. ACM, 2021, pp. 351–361. DOI: 10.1145/3409964.3461806 (cit. on p. 309).

[LNPSY21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. "Vertex connectivity in poly-logarithmic max-flows". In: *STOC*. ACM, 2021, pp. 317–329. DOI: 10.1145/3406325.3451088 (cit. on p. 269).

[LP20] Jason Li and Debmalya Panigrahi. "Deterministic Min-cut in Poly-logarithmic Max-flows". In: *FOCS*. IEEE, 2020, pp. 85–92. DOI: 10.1109/FOCS46700.2020.00017 (cit. on p. 269).

[LPS21] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. "A Nearly Optimal All-Pairs Min-Cuts Algorithm in Simple Graphs". In: *FOCS*. IEEE, 2021, pp. 1124–1134. DOI: 10.1109/FOCS52979.2021.00111 (cit. on p. 269).

[LS14] Yin Tat Lee and Aaron Sidford. "Path Finding Methods for Linear Programming: Solving Linear Programs in Õ(sqrt(rank)) Iterations and Faster Algorithms for Maximum Flow". In: *FOCS*. 2014, pp. 424–433. DOI: 10.1109/FOCS.2014.52 (cit. on p. 269).

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. "A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization". In: *FOCS*. IEEE Computer Society, 2015, pp. 1049–1065. DOI: 10.1109/FOCS.2015.68 (cit. on pp. 273, 309).

[LSZ21] Troy Lee, Miklos Santha, and Shengyu Zhang. "Quantum algorithms for graph problems with cut queries". In: *SODA*. SIAM, 2021, pp. 939–958. DOI: 10.1137/1.9781611976465.59 (cit. on p. 309).

[Mad13] Aleksander Madry. "Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back". In: *FOCS*. IEEE Computer Society, 2013, pp. 253–262. DOI: 10.1109/FOCS.2013.35 (cit. on pp. 269, 275).

[Mad16] Aleksander Madry. "Computing maximum flow with augmenting electrical flows". In: *FOCS*. IEEE. 2016, pp. 593–602. DOI: 10.1109/FOCS.2016.70 (cit. on pp. 269, 275).

[Mas72] John H Mason. "On a class of matroids arising from paths in graphs". In: *Proceedings of the London Mathematical Society* 3.1 (1972), pp. 55–74. DOI: 10.1112/plms/s3-25.1.55 (cit. on pp. 277, 306).

[MN20] Sagnik Mukhopadhyay and Danupon Nanongkai. "Weighted min-cut: sequential, cut-query, and streaming algorithms". In: *STOC*. ACM, 2020, pp. 496–509. DOI: 10.1145/3357713.3384334 (cit. on p. 309).

[Ngu19] Huy L. Nguyen. "A note on Cunningham's algorithm for matroid intersection". In: *CoRR* abs/1904.04129 (2019) (cit. on pp. 273, 278, 279).

[NR72] Jürg Nievergelt and Edward M. Reingold. "Binary Search Trees of Bounded Balance". In: *STOC*. ACM, 1972, pp. 137–142. DOI: 10.1145/800152.804906 (cit. on p. 317).

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. "Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2 - \epsilon})$-time". In: *STOC*. ACM, 2017, pp. 1122–1129. DOI: 10.1145/3055399.3055447 (cit. on p. 272).

[NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. "Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time". In: *FOCS*. IEEE Computer Society, 2017, pp. 950–961. DOI: 10.1109/FOCS.2017.92 (cit. on p. 272).

[Per68] Hazel Perfect. "Applications of Menger's graph theorem". In: *Journal of Mathematical Analysis and Applications* 22.1 (1968), pp. 96–111. DOI: 10.1016/0022-247X(68)90163-7 (cit. on pp. 277, 306).

[Pri15] Christopher Price. "Combinatorial Algorithms for Submodular Function Minimization and Related Problems". MA thesis. University of Waterloo, 2015 (cit. on pp. 283, 284).

[Qua23] Kent Quanrud. "Faster exact and approximation algorithms for packing and covering matroids via push-relabel". In: *CoRR* abs/2303.01478 (2023). DOI: 10.48550/arXiv.2303.01478. arXiv: 2303.01478 (cit. on pp. 269, 276).

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. "Computing Exact Minimum Cuts Without Knowing the Graph". In: *ITCS*. Vol. 94. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 39:1–39:16. DOI: 10.4230/LIPIcs.ITCS.2018.39 (cit. on p. 309).

[RT85] James Roskind and Robert E. Tarjan. "A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees". In: *Math. Oper. Res.* 10.4 (1985), pp. 701–708. DOI: 10.1287/moor.10.4.701 (cit. on p. 271).

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003 (cit. on pp. 269, 270, 298, 300).

[Sha55] Claude E Shannon. "Game playing machines". In: *Journal of the Franklin Institute* 260.6 (1955), pp. 447–453. DOI: 10.1016/0016-0032(55)90186-1 (cit. on p. 274).

[Tar76] Robert Endre Tarjan. "Edge-Disjoint Spanning Trees and Depth-First Search". In: *Acta Informatica* 6 (1976), pp. 171–185. DOI: 10.1007/BF00268499 (cit. on p. 308).

[Wel76] Dominic JA Welsh. *Matroid theory*. Academic Press, London, New York, 1976 (cit. on p. 275).

[Whi88] Walter Whiteley. "The Union of Matroids and the Rigidity of Frameworks". In: *SIAM J. Discret. Math.* 1.2 (1988), pp. 237–255. DOI: 10.1137/0401025 (cit. on pp. 269, 319).

[Wul17] Christian Wulff-Nilsen. "Fully-dynamic minimum spanning forest with improved worst-case update time". In: *STOC*. ACM, 2017, pp. 1130–1143. DOI: 10.1145/3055399.3055415 (cit. on p. 272).

[XG94] Ying Xu and Harold N. Gabow. "Fast Algorithms for Transversal Matroid Intersection Problems". In: *ISAAC*. Vol. 834. Lecture Notes in Computer Science. Springer, 1994, pp. 625–633. DOI: 10.1007/3-540-58325-4_231 (cit. on pp. 269, 271, 274).

Paper F

Nearly Optimal Communication and Query Complexity of Bipartite Matching

Joakim Blikstad, Jan van den Brand,
Yuval Efron, Sagnik Mukhopadhyay,
Danupon Nanongkai

Article published in 63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022 [BBEMN22]
Full version at `https://arxiv.org/abs/2208.02526`.

Abstract

We settle the complexities of the maximum-cardinality bipartite matching problem (BMM) up to poly-logarithmic factors in five models of computation: the two-party communication, AND query, OR query, XOR query, and quantum edge query models. Our results answer open problems that have been raised repeatedly since at least three decades ago [Hajnal, Maass, and Turan STOC'88; Ivanyos, Klauck, Lee, Santha, and de Wolf FSTTCS'12; Dobzinski, Nisan, and Oren STOC'14; Nisan SODA'21] and tighten the lower bounds shown by Beniamini and Nisan [STOC'21] and Zhang [ICALP'04]. We also settle the communication complexity of the generalizations of BMM, such as maximum-cost bipartite b-matching and transshipment; and the query complexity of unique bipartite perfect matching (answering an open question by Beniamini [2022]). Our algorithms and lower bounds follow from simple applications of known techniques such as cutting planes methods and set disjointness.

F.1 Introduction

In the *maximum-cardinality bipartite matching* problem (BMM), we are given a bipartite graph $G = (L \cup R, E)$ with n vertices on each side and m edges. The goal is to find a matching of maximum size in G. This problem, along with its special case of *bipartite perfect matching* (BPM), are central problems in graph theory, economics, and computer science. They have been studied in various computational models such as the sequential, two-party communication, query, and streaming settings [See FF56; HK73; Lov79; KUW85; KVV90; MS04; Zha04; IKLSW12; Mad13; Mad16; GO16; GG17; BHR19; DNO19; AV20; AK20; JST20; BLNPSSSW20; Nis21; AR20b; CKPSSY21; FGT21; AB21; ALT21; FGLPSY21; RSW22; CKLPPS22, and many more]. In this paper, we present simple algorithms and lower bound arguments that settle (up to polylog factors) the complexities of BMM and its generalizations (e.g. max-cost matching and transshipment) in at least five models of computation. Our results answer open problems that have been raised repeatedly since at least three decades ago (e.g. [HMT88; Zha04; IKLSW12; DNO19; Nis21; Ben22a]); see Table F.1 for a summary of our results.

Communication complexity. To be concrete, we start with the two-party communication model, where edges of the input graph G are partitioned between two players Alice and Bob. The goal is for Alice and Bob to compute the value of the BMM or to decide if a BPM exists in G by communicating as frugally as possible. Many fundamental graph problems have been studied in this model since the 80s (e.g. [PS82; BFS86; HMT88; DP89]). For BMM and BPM, their communication complexities have been extensively studied from several angles and perspectives, including exact solution protocols [BFS86; HMT88; IKLSW12; DNO19], round restricted protocols [FKMSZ05; GKK12; GO16; AKL17; AB19b; AR20b], multi-party protocols [GO16; HRVZ15; AKLY16; Kap21; KMT21; HRVZ20], approximate solution protocols [Kap21; KKS14; KKS14; KMNT20; AB21], matrix rank and polynomial representation [Ben22b; BN21], and economics and combinatorial auctions [Rot82; Ten02; Ber09]. In particular, Hajnal, Maass, and Turán [HMT88] showed a lower bound of $\Omega(n \log n)$ for deterministic protocols[1]. For randomized and quantum protocols, the lower bounds are $\Omega(n)$ [BFS86; IKLSW12; Raz92][2]. For an upper bound, Ivanyos, Klauck, Lee, Santha, and de Wolf [IKLSW12] implemented the Hopcroft-Karp algorithm [HK73] to get an $O(n^{3/2} \log n)$-bit deterministic protocol (see also [DNO19; Nis21]).

Closing the large gap between existing upper and lower bounds has been mentioned as an open problem in, e.g., [HMT88; IKLSW12; DNO19; Nis21]. Beniamini

[1] [HMT88] did, in fact, show this lower bound for *st*-connectivity, which, together with folklore reductions, imply the same bound for BPM.

[2] The $\Omega(n)$ lower bound follows by a simple reduction from set-disjointness. [HRVZ20] has shown a $\Omega(\alpha^2 nk)$ lower bound for k-party point-to-point communication model for α-approximation of BMM. [IKLSW12] shows a $\Omega(n)$ quantum communication lower bound by a reduction from inner-product in $\mathbb{F}_2$.

and Nisan [BN21] recently showed that the rank of the communication matrix is $2^{O(n \log n)}$, suggesting that a better upper bound might exist. On the other hand, $\Omega(n^2)$ lower bounds for $o(\sqrt{\log n})$-round communication may suggest that an $\Omega(n^{1+\Omega(1)})$ communication lower bound may exist [FKMSZ05; GKK12; AR20b; CKPSSY21]. In this paper, we resolve this open problem with an $O(n \log^2 n)$ upper bound:

Theorem F.1.1. *The deterministic two-party communication complexity of* BMM *is* $O(n \log^2 n)$.

Note that our protocol can find the actual BMM (Alice and Bob know edges in the BMM in the end) and not just its value. We can in fact solve a more general problem of min-cost bipartite perfect b-matching which implies upper bounds for a large class of problems due to existing reductions (see [BLNPSSSW20]).

Theorem F.1.2. *Given that all the weights/costs/capacities are integers polynomially large in* n*, we can solve the following problems in the two-party edge-partition communication setting, using* $O(n \log^2 n)$ *bits of communication: Maximum-cost bipartite perfect* b*-matching, Maximum-cost bipartite* b*-matching, Vertex-capacitated minimum-cost* (s, t)*-flow, Transshipment (a.k.a. uncapacitated minimum-cost flow), Negative-weight single source shortest path, Minimum mean cycle and Deterministic Markov Decision Process (MDP).*

Models	**Previous papers**		**This paper**
	Lower bounds	Upper bounds	
Two-party communication	$\Omega(n)$ Rand, $\Omega(n \log n)$ Det, Footnote 1 and 2	$\tilde{O}(n^{1.5})$ [DNO19; IKLSW12]	$O(n \log^2 n)$, Det Theorem F.1.1
Quantum edge query	$\Omega(n^{1.5})$ [Zha04; Ben22b]	$O(n^{1.75})$ [LL15]	$\tilde{O}(n^{1.5})$ Theorem F.1.3
OR-query	$\Omega(n)$ Rand, $\Omega(n \log n)$ Det, [BN21]	$\tilde{O}(n^{1.5})$ Det, [Nis21]	$O(n \log^2 n)$, Det Theorem F.1.3
XOR-query	$\Omega(n)$ Rand $\Omega(n^2)$ Det [BN21]	$\tilde{O}(n^{1.5})$ Rand Lemma F.2.14 and [Nis21]	$O(n \log^2 n)$, Rand Theorem F.1.3
AND-query	$\Omega(n)$ Rand, $\Omega(n^2)$ Det [BN21]	$O(n^2)$ Trivial	$\Omega(n^2)$, Rand Theorem F.1.3

Table F.1: The communication and query complexity bounds for BMM and BPM. All upper bounds are stated for BMM and all lower bounds are stated for BPM.

Query complexity. Besides the communication complexity, we also settle the query complexity of BMM and BPM for several variants of the edge query model. In the standard edge query model, the querier can ask whether an edge in the input graph G is present or not. The goal is to solve the graph problem by making as

few queries as possible. This query model, in both deterministic and randomized settings, has been studied for almost half a century [Ros73; RV75; RV76; KSS84; Haj91; CK07] for various graph problems. For BPM, Yao [Yao88] showed that n^2 edge queries are necessary in the deterministic setting and Dürr, Heiligman, Høyer, and Mhalla [DHHM06] showed an $\Omega(n^2)$ lower bounds for the randomized setting[3] thereby completely characterizing (up to constant factors) classical edge query complexity for BPM.

However, for several variants of the classical edge query complexity, there are known gaps between the best known upper and lower bounds for BPM. For example, in the case of *quantum* edge query protocols, Zhang [Zha04] showed a lower bound of $\Omega(n^{1.5})$ by using Ambainis' adversary method [Amb02] (see [Ben22b] for an alternative proof via approximate degree). The best upper bound is, however, at $O(n^{1.75})$ as shown by [LL15]. This upper bound is obtained by simulating the Hopcroft-Karp algorithm using *bomb queries* and relating it to the quantum edge queries.

Another well-studied variant of the classical query protocols is the XOR-query protocols (otherwise known as the *parity decision trees*) where the querier is allowed to ask the following question about the input graph $G = (V, E)$: Given a set S of potential edges of G, is $|S \cap E|$ odd or even? Similarly, AND-queries and OR-queries ask if $S \subseteq E$ or not and if $|S \cap E| \geq 1$ or not, respectively. Such query models have proven to be extremely important in the study of XOR-functions, the log-rank conjecture and lifting theorems [KM93; MO09; CKLM18; HHL18; MS20]. As usual, these query models can be studied in deterministic, randomized and quantum models as well. For graph problems, these query models have recently started to receive increasing attention [BN21; Ben22a; ACK21]. For AND-query or XOR-query complexity, a recent result of [BN21] showed that $\Omega(n^2)$ queries are necessary to compute BPM deterministically[4]. For OR-query, [BN21] also showed a deterministic lower bound of $\Omega(n \log n)$. The upper bound of $\tilde{O}(n^{1.5})$ for OR-queries (and, thereby, *randomized* XOR-queries, see Lemma F.2.14) can be achieved by simulating the Hopcroft-Karp algorithm [Nis21].

From the above results, it remained open to close the polynomial gaps for quantum and OR-queries (as mentioned in [Nis21; Ben22b]) and whether randomization helps for XOR-queries and AND-queries. In this paper, we answer these questions: We provide upper bounds that are tight up to polylogarithmic factors for quantum and OR-queries. Our upper bound result also shows that randomization helps for XOR-queries. In contrast, for AND-queries we can show that an $\Omega(n^2)$ lower bound holds even for randomized algorithms. Our results are summarized below and in Table F.1. Note that our lower bound argument also gives simplified proofs of the lower bounds for XOR-queries and OR-queries.

[3] [Yao88] showed a stronger result: Any non-trivial monotone graph property needs n^2 queries. [DHHM06] mentioned a $\Omega(n^2)$ randomized query complexity for Connectivity. A similar construction (which is essentially a reduction from the query complexity of OR_{n^2}) shows an $\Omega(n^2)$ lower bound for (s,t)-Reachability which reduces to BPM.

[4] For XOR-queries, [BN21] showed that BPM is *evasive*, i.e., requires n^2 queries.

Theorem F.1.3. *The following query bounds hold for* BMM*:*

- *The quantum edge query complexity is* $O(n^{1.5}\log^2 n)$*,*
- *The deterministic* OR*-query complexity is* $O(n\log^2 n)$*,*
- *The randomized* XOR*-query complexity is* $O(n\log^2 n)$*,*

Moreover, the randomized AND*-query complexity of* BPM *is* $\Omega(n^2)$*.*

Finally, our results also extend to the unique bipartite perfect matching problem (UBPM), which has been studied in, e.g., the sequential and parallel settings [KVV85; GKT01; HMT06; Ben22a]. Beniamini [Ben22a] recently show UBPM lower bounds similar to those for BMM and BPM, i.e. $\Omega(n\log n)$ communication complexity, $\tilde{\Omega}(n^{1.5})$ quantum edge query (under a believable conjecture[5]), $\Omega(n\log n)$ OR-queries, $\Omega(n^2)$ XOR-queries, and $\Omega(n^2)$ AND-queries. We complement these lower bounds with tight upper bounds, i.e. $O(n\log^2 n)$ deterministic communication protocol, $O(n^{1.5}\log^2 n)$ quantum edge query algorithm, $O(n\log^2 n)$ deterministic OR-query and randomized XOR-query algorithms, and $\Omega(n^2)$ randomized AND-query lower bound. Our upper bounds answer an open problem by Beniamini [Ben22a].

Update: After our paper was accepted in FOCS 2022, we observed that our technique also leads to a $O(n\log^2 n)$ deterministic protocol in the well-studied *Indenpendent set* (IS) query model [BHRRS18; AL21; RWZ20; AA05; ABKRS04; AB19a]. In this model, a query consists of two disjoint subsets of vertices X and Y, and the answer to the query is 1 iff there is an edge between X and Y (i.e., $E\cap(X\times Y)\neq\emptyset$).

Organization. In Section F.1.1, we provide a brief technical overview of our upper and lower bounds. In Section F.1.2, we list a few open problems that naturally arise from our work. Section F.2 details our various upper bounds, starting with OR-query protocols. In Section F.2.3, we show the applications of the OR-query algorithm, namely two party communication complexity (Section F.2.3), randomized XOR-query (Section F.2.3), Independent set query (Section F.2.3), OR_k-query (Section F.2.3) and quantum edge query (Section F.2.3). We then list different variants of the bipartite matching problem (Section F.3) that our technique can solve as well. Finally, in Section F.4, we provide lower bounds for solving BPM in OR-, AND- and XOR-query settings.

F.1.1 Technical Overview

Upper bounds. Our algorithms follow an existing continuous optimization method. There are many such methods and the question is: *what is the right method?* An intuitive idea would be to implement some fast sequential algorithms for BMM and related problems (e.g. [DS08; Mad13; LS14; Mad16; CMSV17; CLS19; Bra20; LS20; AMV20; BLSS20; BLNPSSSW20; BLLSSSW21; CKLPPS22]), which

[5] [Ben22a] conjectured that the approximate degree of UBPM is $\Omega(n^{1.5})$ (see Conjecture 1) which would imply a similar lower bound for quantum edge query complexity.

are based on *central path methods*. It is not clear, however, how to implement central path methods efficiently in query or communication settings. They require polynomially many iterations (e.g. $\Omega(\sqrt{n})$), each of which needs a large communication and query complexity (e.g. $\Omega(n)$ per iteration). Another option is to use one of the *cutting planes methods* (e.g. the Ellipsoid method). These methods are a framework for solving general convex optimization problems and thus are rather slow for BMM in the sequential setting (e.g. $\tilde{O}(mn)$ time [LSW15]) compared to more specialized alternatives based on central path methods. However, it turns out that cutting planes methods are the right framework for the communication and query settings! In particular, we can implement a cutting planes method with a low number of iterations, such as the *center-of-gravity* (CG) and *volumetric center* (VC) methods [Lev65; New65; Vai89], on the *dual linear program*, i.e. the minimum vertex cover linear program[6]. (We cannot use the Ellipsoid method due to its high number of iterations.) The CG and VC methods are not useful for solving BMM in the sequential setting due to their high running time (the CG method even requires exponential time); however, this high running time is hidden in the internal computation and thus does not affect the communication/query complexities.

Using the cutting planes methods above, our algorithm is simply the following: We start with an assignment $p : V \to \mathbb{R}^+$ on the vertices that is supposed to be a fractional vertex cover of value F, i.e. for every edge (u, v), $p(u) + p(v) \geq 1$ and $\sum_{v \in V(G)} p(v) \leq F$. In each iteration, we need to find a *violated constraint*, i.e. an edge (u, v) such that $p(u) + p(v) < 1$, or the value constraint if $\sum_{v \in V(G)} p(v) > F$. This violated constraint then allows us to compute a new assignment $p : V \to \mathbb{R}^+$ (which is the center of gravity of some polytope) to be used in the next iteration. It can be shown that this process needs to repeat only for $\tilde{O}(n)$ times to construct a fractional vertex cover of value at least F, or conclude no such cover exists.

This simple algorithm leads to efficient algorithms in many settings. For example, in the two-party communication setting, Alice and Bob only need to communicate one violated constraint in each iteration while they can compute the new assignment $p : V \to \mathbb{R}^+$ without any additional communication ($p : V \to \mathbb{R}^+$ depends only on the discovered violated constraints and not on the input graph). It is also not hard to implement this method in other settings. We note that in this paper we use the CG method for simplicity. This method leads to exponential internal computation. This can be made polynomial by using the VC method [Vai89] instead.

Lower bounds. For lower bounds, our goal is to prove a lower bound for BPM (which also implies a lower bound for BMM). Let us start with our randomized AND-query lower bound of $\Omega(n^2)$. A typical approach to show this is proving an $\Omega(n^2)$

[6]We thank an anonymous FOCS'22 reviewer for pointing out the result in [VWW20] that uses the cutting plane method to solve general linear program in the multiparty model of communication. Our result is independent and follows the same general cutting plane framework but we exploit that for our specific linear program, (i) cutting planes have short description and (ii) we have a better bound on the number of iterations.

communication complexity lower bound in the setting defined earlier; however, we have already shown in Theorem F.1.1 that this is not possible. [BN21] sidestepped this obstacle by considering the real polynomial associated with BPM. Known connections between the monomial complexity of this polynomial and AND-query complexity yield corresponding tight $\Omega(n^2)$ *deterministic* AND-query complexity for BPM.

It turns out that we can prove a *randomized* AND-query lower bound (and simplifying the lower bounds proofs of [BN21]) by revisiting the two-party communication lower bounds, but with a slightly different definition. Our main observation here is that AND-queries can be simulated cheaply by the following variant of the two-party communication model: Alice gets edge set $E_A \subseteq E$, Bob gets edge set $E_B \subseteq E$, and they solve BPM (or any other graph function) in the graph $G_\cap = (V, E_A \cap E_B)$. Our AND-query lower bound now follows from a reduction from the set disjointness problem.

Similarly, Beniamini and Nisan [BN21] use real polynomial techniques to prove deterministic lower bounds for XOR-queries and OR-queries. We provide simple alternative proofs via the communication complexity of BMM in the symmetric difference and union graphs $G = (V, E_A \oplus E_B)$ and $G = (V, E_A \cup E_B)$; such lower bounds can be proved via a reduction from the equality and st-reachability problems. Finally note that even though we simplify the query lower bounds proofs, [BN21; Ben22b] showed something stronger, i.e., a complete characterization of the unique multilinear polynomial over reals representing BPM which may have other interesting consequences beyond query complexity.

F.1.2 Open problems

The communication complexity of BMM and BPM has been a bottleneck for many tasks. The fact that it can be solved by a simple cutting planes method might be the gateway to solving many other problems. Below we list some of these problems.

1. **Demand query complexity of BMM.** The demand query setting is equivalent to when we can issue an OR-query only on the edges incident on a single left vertex (or, equivalently, an IS-query where set $X \subseteq L$ is singleton). Minimizing the number of demand queries used to solve BMM and BPM is motivated by economic questions [Nis21; Ben22b]. Like in many settings we consider, the best demand query upper and lower bounds for BMM and BPM are $O(n^{1.5})$ and $\Omega(n)$ respectively. Closing this gap remains open. Because of our efficient IS-query protocol, we believe that a possible direction is to extend our approach to get a better upper bound for demand query. For a better lower bound, our results suggest that one might need a technique specialized for the demand query lower bound: the two known approaches for proving a demand query lower bounds are via quantum and OR-queries (see, e.g., Figure 7 in [Ben22b]) and our quantum and OR-query upper bounds show that these approaches cannot be used.

2. **Bounded communication rounds and streaming passes.** Most graph problems, including BMM and BPM, admit an $\Omega(n^2)$ communication lower bound when only Alice can send a message (i.e. the one-way communication setting) [FKMSZ05]. If Bob gets to speak back once (the 2-round setting), some problems become much easier (e.g. the communication complexity of global edge connectivity reduces from $\Omega(n^2)$ to $\tilde{O}(n)$) [AD21]. Unfortunately, such an efficient protocol for BPM does not exist even when we allow $o(\sqrt{\log n})$ rounds [CKPSSY21; AR20b]. More generally, r-round protocols are known to require $n^{1+\Omega(1/r)}$ communication [AR20b; GO16]. An important question is to get tight r-round communication bounds for BMM and BPM. Our algorithm provides an $\tilde{O}(n)$ communication bound for the extreme case where $r = n$. One possible extension is to study *bounded-iteration* cutting planes methods. For example, can we reduce the number of iterations if in each iteration we can identify more violating constraints? It will be exciting if a polylog(n)-round $\tilde{O}(n)$-communication protocol exists. It will be even more exciting if this can be extended to a polylog(n)*-passes streaming algorithm* (breaking [LSZ20; AJJST22] and matching [GO16]).

3. **Distributed Matching.** The distributed CONGEST model is an important model to study fundamental graph problems (e.g. minimum spanning tree, shortest paths, and minimum cut) on distributed networks (e.g. [GHS83; KP98; Nan14; GL18; FN18; Elk20; BN19; AR20a; GKKLP18; HKN21; CM20; GNT20; NS14; DEMN21]). Compared to other graph problems, computing BMM and BPM exactly in CONGEST is much less understood in this model. This is despite the studies of their variants since the 80s [Lub86; II86; Gal16; AKO18; AK20]. The best lower bound for this problem is $\tilde{\Omega}(\sqrt{n} + D)$[AKO18; DHKKNPPW12] (see also [HWZ21]). The best upper bound is $O(n \log n)$ [AKO18]. For sparse graphs, the upper bound can be improved to $\tilde{O}(m^{3/7}(\sqrt{n}D^{1/4} + D))$ via continuous optimization [FGLPSY21]. (Better upper bounds via fast matrix multiplication also exist on the special case of *congested clique* [Gal16].) A major open problem is to close the gap between upper and lower bounds. Our results may suggest a new approach for improving the known upper bounds for the problem. Past results seem to suggest that graph problems with $\tilde{O}(n)$ communication complexity usually admit an $\tilde{O}(\sqrt{n} + D)$ upper bound in CONGEST. (A recent example is the $\tilde{O}(n)$ communication complexity protocol of mincut [MN20] that was later extended to achieve an $\tilde{O}(\sqrt{n} + D)$ upper bound in CONGEST [DEMN21].) Proving that this is or is not the case for BMM and BPM will be an exciting result.

4. **General Matching.** The maximum matching problem on *general* (i.e. not-necessarily-bipartite) graphs is less understood than that on bipartite graphs. Unlike BMM, the linear programming formulations for general matching is rather unwieldy, making it difficult to apply the cutting planes method approach. Settling the communication and query complexity of general matching

remain intriguing open problems. On one hand, there might be a hope to show truly super-linear (i.e., $\Omega(n^{1+\epsilon})$ for some constant $\epsilon > 0$) communication lower bounds in these models, thereby showing a gap between the bipartite and non-bipartite case. On the other hand, an $\tilde{O}(n)$ communication complexity upper bound for the general matching problem would hopefully shed some light on the interplay between matchings on bipartite versus general graphs.

5. **Maxflow/mincut and Related Problems.** Max (s,t)-flow, equivalently min (s,t)-cut, is a powerful tool that can be used to solve BMM, BPM, and many other fundamental graph problems. Efficiently solving this problem could only be a dream in the past in many computational models since even its special case of matching could not be solved efficiently. Our results serve as a step toward this goal. Particularly interesting goals are solving (s,t)-max-flow/min-cut in the communication[7], distributed, cut query, and streaming settings (Bounded round communication lower bounds in multiparty communication setting for (s,t)-max-flow/min-cut have been studied in [ACK19].). Also, there are problems that were recently shown to be solvable in max-flow time in the sequential setting such as Gomory-Hu tree, vertex connectivity, Steiner cut, hypergraph global min-cut, and edge connectivity augmentation [CGLNPS20; LP20; LP21; LNPSY21; CQ21; MN21]. Can these problems be solved as efficiently as max-flow in other settings, e.g. the communication, distributed, and streaming settings?

Other problems include (i) showing $\Omega(n \log n)$ *randomized* communication lower bound for connectivity or even just for BMM and min-cost flow, (ii) closing the $\log n$ factor gap between OR-query upper and lower bounds, and (iii) settling the quantum OR-query and AND-query complexity of BMM and BPM.

F.2 Bipartite Matching Upper Bounds

Our goal in this section is to present a simple OR-query algorithm based on the cutting planes framework to find a maximum matching of a bipartite graph, i.e. to solve the BMM problem. From there we show how our OR-query algorithm can be translated to several other information theoretical models of computation. Formally, the following is the main theorem of the section.

Theorem F.2.1. *Given n, there are algorithms solving* BMM *in the following models.*

1. *Deterministic two-party edge-partition communication, with communication complexity $O(n \log^2 n)$.*

[7] Here we expect Alice and Bob to know the flow values in their respective sets of edges. The decision version of this problem where we ask if the total flow is at least a threshold k is also interesting.

2. *Deterministic* OR*-query, with query complexity* $O(n \log^2 n)$.
3. *Randomized* XOR*-query, with query complexity* $O(n \log^2 n)$.
4. *Quantum edge query, with query complexity* $O(n^{1.5} \log^2 n)$.

Overview. We employ a standard cutting planes framework to determine if a bipartite graph has a vertex cover of a given size F or not. We show that this cutting planes method can be implemented in $O(n \log n)$ iterations, where in each iteration we access the input graph a small number of times ($O(\log n)$) using OR-queries to find an edge that corresponds to a violated constraint (i.e. a cutting plane), if one exists. Throughout this work, we use the following well known characterization of the existence of a matching of a certain size in a bipartite graph.

Claim F.2.2 (König's Theorem)**.** *A bipartite graph G has a minimum vertex cover of size F if and only if it does not have a matching of size $F+1$.*

The vertex cover linear program. For a bipartite graph $G = (V, E)$ with $V = L \cup R$, $|L| = |R| = n$, the following linear program ($\mathcal{P}^G$) over $x \in \mathbb{R}^V$ describes the fractional minimum vertex cover problem on G. Since G is bipartite, the constraint matrix is totally unimodular, and hence ($\mathcal{P}^G$) is integral [KVKV11, Section 5], i.e. there exists an integer optimal solution to ($\mathcal{P}^G$).

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \forall (u,v) \in E \\ & 0 \leq x_v \leq 1 & \forall v \in V \end{array} \tag{$\mathcal{P}^G$}$$

Decision version. We first consider a *decision version* of our problem, namely given an integer F we want to determine if G has a matching of size at least $F+1$. Note that if we can solve this decision version, then we can also—by binary-searching over F—solve the *optimization version* (i.e. finding the minimum size of a vertex cover / maximum size of a bipartite matching) with an overhead of $O(\log n)$. We start by focusing on solving the decision version (Section F.2.1), and later (in Section F.2.2) we show how to, via a simple modification of the algorithm, actually solve the optimization version *without* this extra $O(\log n)$ binary-search overhead.

By König's Theorem (Claim F.2.2), determining whether G has a matching of size at least $F+1$ is equivalent to determining whether G (does not) have a vertex cover of size at most F. This is equivalent to determining if ($\mathcal{P}^G$) has some feasible solution x with $\sum x_v \leq F$. So we define another polytope ($\mathcal{P}^G_F$) as follows:

$$\begin{array}{lll} \sum_{v \in V} x_v \leq F + \frac{1}{3} & & \\ x_u + x_v \geq 1 & \forall (u,v) \in E & \\ 0 \leq x_v \leq 1 & \forall v \in V & \end{array} \tag{$\mathcal{P}^G_F$}$$

Our decision algorithm either finds a feasible point for the above polytope, or it finds a witness of $\mathcal{P}_F^G$ having no feasible points in the form of a set of edges that contains a matching of size $F+1$.

Note that we relax the constraint $\sum x_v \leq F$ a bit to $\sum x_v \leq F + \frac{1}{3}$. This ensures that our polytope has a significantly large volume if it is non-empty (see Lemma F.2.3). Thus our cutting planes methods can terminate and conclude that the polytope is empty whenever the volume is too small. This relaxation does not impact the correctness of our algorithm: since $(\mathcal{P}^G)$ is integral, it has an integral optimal objective value, which means that if a feasible solution x of $(\mathcal{P}_F^G)$ exists, then there also exists a feasible solution x' which achieves $\sum x'_v \leq F$.

Lemma F.2.3. *For any bipartite graph $G = (V, E)$, if F is an integer such that $(\mathcal{P}_F^G)$ is non-empty, then* $\mathrm{vol}(\mathcal{P}_F^G) \geq \left(\frac{1}{20n}\right)^{2n}$.

Proof. Let x be an integral solution for $(\mathcal{P}_F^G)$, of value F. Indeed, if $(\mathcal{P}_F^G)$ is feasible, then such an x must exist due to the integrality of $(\mathcal{P}^G)$. Let $I_0 = \{i \in [2n] \mid x_i = 0\}$ and $I_1 = \{i \in [2n] \mid x_i = 1\}$. We argue that the hypercube $[\frac{1}{20n}, \frac{1}{10n}]^{I_0} \times [1 - \frac{1}{20n}, 1]^{I_1}$ is completely contained in $(\mathcal{P}_F^G)$. That is, if, for each x_i with $x_i = 1$ we replace it with any value in $[1 - \frac{1}{20n}, 1]$; and for each x_i with $x_i = 0$ we replace it with with any value in $[\frac{1}{20n}, \frac{1}{10n}]$; the point remains feasible for $(\mathcal{P}_F^G)$. We verify this below.

- The $0 \leq x_v \leq 1$ constraints remain valid.
- Similarly, the $\sum x_v \leq F + \frac{1}{3}$ constraint remains valid, since we increase the value of x_i by at most $\frac{1}{10n}$ for each i, and there are $2n$ vertices in total (so we increase $\sum x_v$ by at most $\frac{1}{5}$).
- Lastly, the constraint $x_u + x_v \geq 1$ (for an edge $(u, v) \in E$) also remains valid, as either (i) both x_u and x_v were 1 before, in which case we now have $x_u + x_v \geq 2 - \frac{1}{10n}$; or (ii) exactly one of x_u or x_v was 1 before, in which case we increased the variable which was 0 by at least $\frac{1}{20n}$ and decreased the variable which was 1 by at most $\frac{1}{20n}$.

Thus we have argued that a hypercube of volume $\left(\frac{1}{20n}\right)^{2n}$ is contained in $(\mathcal{P}_F^G)$. □

F.2.1 OR-query decision algorithm

In this section we describe our cutting planes based OR-query algorithm for solving the feasibility problem on $(\mathcal{P}_F^G)$. We begin with a verbal overview of the algorithm, followed by pseudocode in Algorithm F.1. The main lemma of this section is the following.

Lemma F.2.4. *Given an integer F, there is a deterministic algorithm (Algorithm F.1) using $O(n \log^2 n)$ OR-queries which on an input bipartite graph $G = (V, E)$ either finds a feasible point in $(\mathcal{P}_F^G)$, or else a witness, in the form of a matching of size $F+1$, that $(\mathcal{P}_F^G)$ is empty.*

Center-of-gravity cutting planes method. We are now ready to introduce the cutting planes framework [Lev65; New65]. The idea is that we start with the polyhedra $P_0 = \{x \in [0,1]^V : \sum x_v \leq F + \frac{1}{3}\}$ (which contains $(\mathcal{P}_F^G)$), and repeatedly find "good" constraints "$x_u + x_v \geq 1$" (corresponding to edges $(u,v) \in E$) to add which reduce the volume sufficiently fast. Eventually, we either find a (fractional) feasible solution to $(\mathcal{P}_F^G)$, or have determined that no such feasible point exist.

We work in iterations, each iteration i is characterized by a polyhedron $P_i \supseteq (\mathcal{P}_F^G)$. We compute the *center-of-gravity* of P_i, denoted by $p_i = cg(P_i) \in P_i$, and defined to be $cg(P_i) = \left(\int_{P_i} z \, \mathrm{d}z\right) / \left(\int_{P_i} \mathrm{d}z\right)$. Note that we know P_i, so our algorithm can compute[8] $p_i = cg(P_i)$ without using any queries.

Either p_i is feasible for $(\mathcal{P}_F^G)$, in which case the cutting planes algorithm reports this and terminates. Otherwise there must exist some *violated constraint* "$x_u + x_v \geq 1$" in $(\mathcal{P}_F^G)$ but not in P_i (i.e. p_i does not satisfy this constraint, that is $p_i^u + p_i^v < 1$). In this case, we want to find such a violated constraint, and let $P_{i+1} = P_i \cap \{x \in \mathbb{R}^V : x_u + x_v \geq 1\}$, after which we continue with the next iteration of the cutting planes method on P_{i+1}. We say that an edge $(u,v) \in E$ is a *violating edge* for iteration i if $p_i^u + p_i^v < 1$. The process of finding a violating edge is the only part of the algorithm which requires access to the input graph, and hence the only place where OR-queries are being issued. Essentially, we need to implement a *separation oracle* FindViolatingEdge, which we explain how to do with OR-queries in Claim F.2.5. The full algorithm can be found in Algorithm F.1.

Claim F.2.5 (OR-implementation of FindViolatingEdge)**.** *Using $O(\log n)$ OR-queries we can find a violating edge or else determine that none exist.*

Proof. Given the center-of-gravity point p_i, we let $S = \{(u,v) \in L \times R \mid p_i^u + p_i^v < 1\}$ be the set of pairs of vertices (u,v) which would be a violating edge if this pair was also an edge of the graph. Our task is thus to find some edge $e \in S \cap E$, or else determine that $S \cap E$ is empty. This can be done by a binary-search (with OR-queries) over S. □

We now turn to prove several properties about our Algorithm F.1.

Observation F.2.6. *Let i be some iteration of the execution of Algorithm F.1, then $P_i \supseteq \mathcal{P}_F^G$.*

Proof. For every i, the set of constraints defining P_i is, by the behaviour of the algorithm, a subset of the constraints defining $\mathcal{P}_F^G$, thus the observation follows. □

Lemma F.2.7. *The algorithm terminates after $O(n \log n)$ iterations of the cutting planes method.*

[8] Finding the center-of-gravity in an n-dimensional polyhedron is NP-hard. However, all the considered models in Theorem F.2.1 are query models, and in particular are purely information-theoretical, and we can thus disregard computational concerns. For ease of presentation, we work with center of gravity, but alternatively, one could use other variants of cutting plane using more computationally efficient notions of "center" such as volumetric centers [Vai89].

Algorithm F.1: OR-query algorithm for BMM

Input: OR-query access to $G = (L \cup R, E)$, vertex set $L \cup R$, feasibility parameter F

Output: Whether $(\mathcal{P}_F^G)$ is feasible

$P_0 \leftarrow \left\{ x \in [0,1]^{2n} \;\middle|\; \sum_{v \in V} x_v \leq F + \frac{1}{3} \right\}$
$E' \leftarrow \emptyset$
$i \leftarrow 0$
while $vol(P_i) \geq \left(\frac{1}{20n}\right)^{2n}$ **do**
  $p_i \leftarrow cg(P_i)$
  $(u, v) \leftarrow \mathsf{FindViolatingEdge}(E', p_i)$
  if *no edge was found* **then**
    return "Feasible" // p_i is feasible for $(\mathcal{P}_F^G)$
  $E' \leftarrow E' \cup \{(u, v)\}$
  $P_{i+1} \leftarrow P_i \cap \{x \in \mathbb{R}^{2n} \mid x_u + x_v \geq 1\}$
  $i \leftarrow i + 1$
return "Infeasible" // E' contains a matching of size $F + 1$

Proof. We use the following well-known property of the center of gravity of a convex polytope.

Lemma F.2.8 ([Grü60])**.** *For any convex polytope P with center of gravity c and any halfspace $H = \{x \mid \langle a, (x - c)\rangle \geq 0\}$ passing through c, it holds that:*

$$\frac{1}{e} \leq \frac{\mathrm{vol}(P \cap H)}{\mathrm{vol}(P)} \leq \left(1 - \frac{1}{e}\right).$$

This implies that, in our case, $\mathrm{vol}(P_{i+1}) \leq (1 - \frac{1}{e})\mathrm{vol}(P_i)$. This means that in each iteration, we either find a feasible solution to $(\mathcal{P}_F^G)$, or cut down the volume by a constant fraction as we have found a violating edge. Initially, $\mathrm{vol}(P_0) \leq 1$, since it is contained in the unit-hypercube $[0,1]^{2n}$. By Lemma F.2.3 we can terminate when P_i has volume less than $\left(\frac{1}{20n}\right)^{2n}$ and conclude that $(\mathcal{P}_F^G)$ is empty in this case. This happens after at most $O\left(\log((20n)^{2n})\right) = O(n \log n)$ iterations. □

Lemma F.2.9. *Let i_{max} denote the last iteration in the execution of the algorithm. Then either $p_{i_{max}} \in \mathcal{P}_F^G$ which serves as a witness that a vertex cover of size F exists, or $\mathcal{P}_F^G = \emptyset$ and the set $E' \subseteq E$ (constructed by the algorithm) contains a matching of size $F + 1$.*

Proof. In the case where we find a feasible point p in $(\mathcal{P}_F^G)$, this point is a fractional vertex cover of size at most $F + \frac{1}{3}$ for our graph (and hence a non-constructive witness that there exists an (integral) vertex cover of size F in the graph).

On the other hand, suppose we determined that $(\mathcal{P}_F^G)$ is empty, which means we got to an iteration i_{max} where $\text{vol}(P_i) < \left(\frac{1}{20n}\right)^{2n}$. We argue that this actually means that the polyhedron P_i is empty. That is, we argue that we have found a set of edges $E' \subseteq E$ which contain a matching of size $F+1$ (E' is the set of edges whose constraints we added to $P_{i_{max}}$ during the cutting planes method). If this was not the case, that is if the maximum matching size in E' is at most F, then it must be the case, by Claim F.2.2, that a vertex cover of size F exists in the subgraph $G' = (L \cup R, E')$, and hence that some integer point exists in our polyhedron $P_{i_{max}}$. We can deduce that this is impossible, however, by simply noting that by the behaviour of the algorithm, it holds that $(\mathcal{P}_F^{G'}) = P_{i_{max}}$, and thus we can apply Lemma F.2.3 which then says that $\text{vol}(P_i) \geq \left(\frac{1}{20n}\right)^{2n}$, which is a contradiction. □

By Claim F.2.5 and Lemma F.2.7 we see that the algorithm makes a total of $O(n \log^2 n)$ OR-queries, and Lemma F.2.9 argues its correctness. This concludes the proof of Lemma F.2.4.

F.2.2 OR-query optimization algorithm

In this section we describe a standard modification (see e.g. [Vai89, Section 4]) to our cutting planes *decision* algorithm, so that it solves the *optimization* version with the same query-complexity.

Lemma F.2.10. *There is a deterministic algorithm using $O(n \log^2 n)$ OR-queries which solves the BMM problem. In particular, the algorithm finds a maximum matching M, together with a witness that M is maximum in the form of a fractional vertex cover of size strictly less than $|M|+1$.*

Proof. The idea is to run Algorithm F.1 starting with $F = 2n$. Whenever the algorithm finds a feasible point p_i, instead of terminating, we lower the value of F instead. The point p_i is a certificate that a vertex cover of size $\lfloor \sum_v p_i^v \rfloor$ exists (since $(\mathcal{P}^G)$ is integral). Hence we lower F to $F \leftarrow \lfloor \sum_v p_i^v \rfloor - 1$, by adding the constraint $\sum_v x_v \leq F + \frac{1}{3}$, and continue the cutting planes algorithm. Note that the constraint $\sum_v x_v \leq F + \frac{1}{3}$ forms a *violating constraint* for p_i (and therefore cuts down the volume by a constant fraction, see Lemma F.2.8, and counts as an iteration of the cutting planes algorithm).

At the end, the algorithm must terminate by determining that $(\mathcal{P}_F^G)$ is empty (for the current value of F), in which case the found edges E' contains a matching of size $F+1$ (see Lemma F.2.9). On the other hand, the last time we lowered F, we had a fractional vertex cover p_i of size strictly less than $F+2$. □

F.2.3 Applications

The goal of this section is to complete the proof of Theorem F.2.1. We prove the theorem by showing how to simulate the OR-query cutting planes algorithm in the

communication setting and the different query models (randomized XOR, IS, OR_k, and quantum edge query).

Communication complexity

We first consider the two-party edge-partition communication setting, where the edges E of the graph are partitioned into sets E_A and E_B given to Alice and Bob respectively.

Claim F.2.11. *There is a communication protocol solving* BMM *in* $O(n \log^2 n)$ *bits of communication.*

A standard way of doing this is to simulate each OR-query $S \subseteq L \times R$ with 2 bits of communication: Alice and Bob check locally if $S \cap E_A$, respectively $S \cap E_B$, is non-empty and then share this information with each-other.

Alternatively, Alice and Bob can implement the "FindViolatingEdge"-subroutine of Algorithm F.1 directly by checking locally for a violating edge and sharing it, if they find one, to the other party. This makes sure that E' is mutually known throughout the protocol. Sending an edge requires $O(\log n)$ bits of communication, and needs to be done $O(n \log n)$ times. So this alternative approach achieves the same final communication complexity (although in slightly fewer rounds of communication), and is also closer to our weighted matching algorithm in Section F.3.2 (where the query-settings are no longer compatible).

Randomized XOR-query

Now we turn to the XOR-query setting. [BN21] showed that solving BPM is *evasive* for the XOR-query setting for any *deterministic* algorithm, meaning that any such algorithm needs to make n^2 queries (that is, the trivial algorithm for querying every potential edge individually is optimal)! Nevertheless, we show that *randomized* XOR-query algorithms are much more powerful, and can achieve almost linear number of queries instead.

Claim F.2.12. *There is a randomized algorithm which makes* $O(n \log^2 n)$ XOR*-queries and, w.h.p.*[9]*, solves* BMM*.*

In order to establish this result, we need the following folklore observation.

Observation F.2.13. *For any* k*, let* $x \in \{0,1\}^k$ *be a binary string of length* k*, such that* $x \neq 0^k$*. If* $r \in \{0,1\}^k$ *is sampled uniformly at random, then* $\Pr\left[(\sum_{i=1}^{k} x_i r_i) \text{ is odd}\right] = \frac{1}{2}$.

Lemma F.2.14. *A single* OR*-query can be simulated, w.h.p., by issuing* $O(\log n)$ *randomized* XOR*-queries.*

[9] *w.h.p. = with high probability*; meaning with probability at least $1 - 1/n^c$ for an arbitrarily large constant c.

Proof. If we want to simulate an OR-query over a subset S, we can sample $S' \subseteq S$ randomly (independently keep every element with probability $\frac{1}{2}$) and issue an XOR-query over S'. If the answer to said OR-query was "YES", then we have, by Observation F.2.13, a constant probability of realizing this with our XOR-query over S'. If we repeat $O(\log n)$ times, we can answer the OR-query correctly w.h.p. □

Proof of Claim F.2.12. Just applying Lemma F.2.14 to our OR-query algorithm would imply an $O(n \log^3 n)$ randomized XOR-query algorithm. An additional observation is required to bring the query complexity down to $O(n \log^2 n)$. We note that in each invocation of FindViolatingEdge, we need only simulate the first OR-query, after which we, w.h.p., have in hand a concrete set $S' \subseteq S$ for which $\mathsf{XOR}(S') = 1$ (or else determined that the answer to said OR-query should be "NO"). At this point we can binary-search *deterministically* using an additional $O(\log n)$ XOR-queries to find a violating edge in S'. Hence, each invocation of FindViolatingEdge can be simulated, w.h.p., via $O(\log n)$ XOR-queries; and thus by Lemmas F.2.7 and F.2.9, the entire algorithm requires $O(n \log^2 n)$ XOR-queries and is correct w.h.p. □

Independent set (IS) query

In this section we discuss a restricted version of the OR-query, namely the *Independent Set* (IS) query, as studied by, for example, [BHRRS18; AL21; RWZ20; AA05; ABKRS04; AB19a]. An IS-query consists of specifying two subsets $X \subseteq L$ and $Y \subseteq R$ and asking if there is any edge between some vertex in X and some vertex in Y (or, conversely if $X \cup Y$ forms an independent set)[10].

Claim F.2.15. *There is a deterministic algorithm that solves* BMM *with* $O(n \log^2 n)$ IS*-queries.*

Proof. In each iteration, the cutting plane method finds some fractional point $p \in \mathbb{R}^{L \cup R}$, and we are asked to implement a separation oracle FindViolatingEdge for this point. That is we want to determine if any edge in the set $S = \{(u, v) \in L \times R \mid p_u + p_v < 1\}$ exists (and if so find it). With unrestricted OR-queries this is easy (see Claim F.2.5), however it might not be the case that this set S is structured like an IS-query. In the case when p is integral, we can define $X = \{v \in L : p_v = 0\}$ and $Y = \{v \in R : p_v = 0\}$, and note that $S = X \times Y$. Hence, in the case of integral p, we can implement FindViolatingEdge using IS-queries: first we binary search on X, and then on Y, to find the violating edge if it exists.

We argue that there always exist an integral point $p' \in \mathbb{Z}^{L \cup R}$ which we can use instead of p when calling the separation oracle FindViolatingEdge. The integral point p' will satisfy the following two properties:

[10] Generally, an Independent set query specifies only one subset of vertices whereas the *Bipartite* independent set query specifies two disjoint sets of vertices as defined here. However, for bipartite graphs, these two types of queries are equivalent.

(i) For all pairs $(u, v) \in L \times R$, if $p_u + p_v \geq 1$ then $p'_u + p'_v \geq 1$ too. This means that if we found a violating edge for p', the same edge is also violating for p.

(ii) $\sum p'_v \leq \sum p_v$. This means that if there were no violating edges (i.e. p' formed a vertex cover), we have found a certificate that the maximum matching size is at most $\sum p'_v \leq \sum p_v$.

Indeed, consider the bipartite graph H with edge set $\{(u, v) \in L \times R : p_u + p_v \geq 1\}$. In H, p is a (fractional) vertex cover of size $\sum p_v$. This means that there exists an integral vertex cover of size $\lfloor \sum p_v \rfloor$ in H, since the minimum vertex cover linear program is integral for bipartite graphs. Therefore, we pick p' to be an arbitrary such integral vertex cover, and we note that by definition it satisfies the above properties (i) and (ii). □

OR_k-query

Here we discuss the OR-query of *limited width* k, i.e. the OR_k-query. That is, we are only allowed to ask OR-queries over sets $S \subseteq L \times R$ of size $|S| \leq k$. This model turns out to be useful as an intermediary step towards proving tight upper bounds for the quantum edge query model (see Section F.2.3). Considering this model also helps to unveil the difficulty behind designing *demand query* algorithms (see open problems in Section F.1.2 for further discussion) for BPM, pointing to the fact that the barrier is not the size of the query, but rather its locality.

Claim F.2.16. *There is a deterministic algorithm that solves* BMM *with* $O(n \log^2 n)$ OR_n*-queries.*

In fact, we show, via an amortization argument, that *any* OR-query algorithm (for an arbitrary graph problem) can be simulated with OR_k-queries with an additive overhead dependent on k.

Lemma F.2.17. *Any* OR*-query algorithm* $\mathcal{A}$ *(for any graph problem) making* q *queries can be converted to an* OR_k*-algorithm making* $q + \lfloor \frac{n^2}{k} \rfloor$ *queries.*

Proof. The main idea of the proof is the following: Every time an OR-query answers "NO", we know that none of the queried edges are present in the graph G. This is an important piece of information that helps us save queries in the future. More formally, we use the following amortization argument.

We keep track of a set $E^c \subseteq L \times R$ of pairs (u, v) which we know are **not** edges of G, that is $E \cap E^c = \emptyset$. Whenever $\mathcal{A}$ issues an OR-query $S \subseteq L \times R$ we do the following. Let $S_1, S_2, \ldots, S_r$ be a partition of $S \setminus E^c$ so that $|S_1| = |S_2| = \ldots |S_{r-1}| = k$, and $|S_r| \leq k$. We issue the OR_k-queries $S_1, S_2, \ldots, S_r$ sequentially, in order, until we get a "YES" answer which we return to $\mathcal{A}$ (or else, after we have received "NO" from all the sets we return a "NO" answer to $\mathcal{A}$). For the last query which we made (which was either a query to S_r, or a query which returned "YES"), we charge the cost to $\mathcal{A}$. In total, we thus charge at most q cost to $\mathcal{A}$: one per OR-query $\mathcal{A}$ issues.

For all the queries to S_i which got "NO" answers and for which $i < r$, we update $E^c \leftarrow E^c \cup S_i$. Hence, for each such "NO" answer we have increased the size of E^c by k. Note that this can happen at most $\lfloor \frac{n^2}{k} \rfloor$ times. So, other than the q queries charged to $\mathcal{A}$, we have made at most $\lfloor \frac{n^2}{k} \rfloor$ queries. Hence, in total, we made $q + \lfloor \frac{n^2}{k} \rfloor$ OR_k-queries to simulate the q many OR-queries from $\mathcal{A}$. □

Plugging in our $O(n \log^2 n)$ OR-query algorithm to the above lemma yields Claim F.2.16.

Quantum edge query (Q_2)

In this section we consider the quantum **edge**-query model. See [BW02] for a formal definition and [NC16] for a more extensive background on quantum computing.

Claim F.2.18. *The quantum edge query complexity of solving* BMM *is* $O(n^{1.5} \log^2 n)$.

We use our OR_k-query algorithm (Lemma F.2.17) together with a well known quantum result (Lemma F.2.19) regarding the quantum query complexity of the OR function.

Lemma F.2.19 (Grover Search [Gro96])**.** *There is quantum query algorithm that computes w.h.p. the* OR *function over* k *bits with query complexity* $O(\sqrt{k} \log k)$.

Proof of Claim F.2.18. Consider the instance of Lemma F.2.17 where we put $k = \frac{n}{\log^2 n}$, then we obtain an OR_k-query algorithm for BMM with $O(n \log^2 n)$ queries. Each such OR_k-query can be simulated, w.h.p., using $O(\sqrt{k} \log n) = O(\sqrt{n})$ quantum edge queries, by Lemma F.2.19. □

F.3 Weighted and Vertex-Capacitated Variants

In this section we show that the cutting planes method is strong enough to be generalized to solve weighted and (vertex-)capacitated problems, for example *max-cost b-matching*. As an application, we also show how to solve *unique bipartite perfect matching* (UBPM) in Section F.3.3. For the weighted problems, we focus on the two-party edge-partition communication setting, since there is no natural generalization of the OR-queries.

Theorem F.3.1. *Given that all the weights/costs/capacities are integers polynomially large in* n*, we can solve the following problems*[11] *in the two-party edge-partition communication setting, using* $O(n \log^2 n)$ *bits of communication.*

(i) Maximum-cost bipartite perfect b-matching.

(ii) Maximum-cost bipartite b-matching.

[11] See [BLNPSSSW20, Section 8.6 (full version)] for a more extensive discussion of these variants.

(iii) Vertex-capacitated minimum-cost (s,t)-flow.

(iv) Transshipment (a.k.a. uncapacitated minimum-cost flow).

(v) Negative-weight single source shortest path.[12]

(vi) Minimum mean cycle.

(vii) Deterministic Markov Decision Process (MDP).

Reductions. Problems (i), (ii), (iii), (iv) are equivalent, and problems (v), (vi), (vii) can all be reduced to, for example, (i). All these reduction are shown in [BLNPSSSW20] (and can be verified as rectangular reductions, i.e., compatible with the two-party communication setting), *except* that (ii) (i.e. max-cost, not-necessarily-perfect, bipartite b-matching) can solve any of (actually really all of): (i), (iii), (iv); which we show in Section F.3.1.

These reductions allow us to focus on a single one of these problems. We pick item (ii), that is *max-cost (not-necessarily-perfect)*[13] *bipartite b-matching*, which is the one which most closely resembles the unweighted bipartite matching problem. In Section F.3.2 we show how the cutting planes framework can be generalized to work with the costs c and demand vector b.

Definition F.3.2 (b-matching)**.** Given a graph $G = (V, E)$, a demand vector $b \in \mathbb{Z}^V_{\geq 0}$, and edge-costs $c \in \mathbb{Z}^E$, we call a vector $y \in \mathbb{Z}^E_{\geq 0}$ a *b-matching* (or a *fractional* b-matching if we allow $y \in \mathbb{R}^E_{\geq 0}$) if $\sum_{e \in \delta(v)} y_e \leq b_v$ for all $v \in V$ (where $\delta(v)$ is the set of edges incident to v). If $\sum_{e \in \delta(v)} y_e = b_v$ for all $v \in V$, then y is a *perfect* b-matching. The *cost* (or *weight*) of y is $\sum_{e \in E} c_e y_e$.

F.3.1 Max-cost perfect b-matching $\rightarrow$ Max-cost b-matching

We can reduce the perfect variant to the not-necessarily-perfect one. Suppose we are given an instance $(G = (V, E), b \in \mathbb{Z}^V_{\geq 0}, c \in \mathbb{Z}^E)$ of the perfect variant which we wish to solve. Firstly, we may assume that the costs are non-negative, since adding a constant W to all costs will increase the cost of a *perfect* b-matching by exactly $W \frac{\sum_{v \in V} b_v}{2}$.

If we just solve max-cost b-matching, we in general do not obtain a perfect b-matching, since matchings of smaller cardinality might have higher cost. To encourage the max-cost b-matching to prioritize perfect matchings over non-perfect matchings, we simply add a large integer W to all the the costs. That is we use the cost function $c'_e = c_e + W$ instead (again, we can do this since we know exactly

[12] Although the reduction shown in [BLNPSSSW20] is randomized, we note that it can be made deterministic by noticing that both parties in the end will know all the edges of a shortest path tree.

[13] Some of the other problems, e.g. *perfect* b-matching, can have unbounded dual linear programs which make them trickier to work with in the cutting planes framework.

how this will affect the cost of a perfect b-matching). If W is sufficiently large (in particular set $W := 1 + |V| \cdot \max c_e$), any max-cost b-matching will also be a perfect b-matching (if one exist).

If it was not, suppose M is a non-perfect b-matching and of maximum cost for c', in a graph which allows a perfect b-matching. Then there must exist an augmenting path in M of length $2\ell+1 \leq |V|$ (where we add $\ell+1$ edges and remove ℓ). The total cost (w.r.t. c') of the added edges is now at least $(\ell+1)W$, while the cost of the removed edges is at most $\ell(W + \max c_e) \leq \ell W + |V| \max c_e < \ell W + W = (\ell+1)W$. That is we added more cost than we removed, hence contradicting that M was of maximum cost.

F.3.2 Cutting planes method for max-cost bipartite b-matching

In this section we briefly explain how the cutting planes algorithm can solve the *max-cost bipartite b-matching*, and hence prove Theorem F.1.2. The details are postponed to Section F.5. The main result of this section is the following:

Lemma F.3.3. *Max-cost bipartite b-matching can be solved using $O(n \log^2(nW))$ communication, where $W := \max\{\max |c_e|, \max b_v\}$ is the largest number in the input.*

Let $G = (V, E)$ (bipartite with $|V| = n$), $b \in \mathbb{Z}^V_{\geq 0}$ and $c \in \mathbb{Z}^E$ be an instance of the max-cost bipartite b-matching problem. We assume the edges (together with their costs) are partitioned between two parties Alice and Bob, say Alice owns E_A (together with c_e for $e \in E_A$) and Bob E_B (together with c_e for $e \in E_B$). We assume both players know the demands b (otherwise it can be communicated in $O(n \log W)$ bits).

Dual linear program. Similarly as for the unweighted bipartite matching problem, we run a cutting planes algorithm on the dual linear program ($\mathcal{P}^{(G,b,c)}$) (refer to the constraints of Definition F.3.2 for the primal linear program). We can think of $x \in (\mathcal{P}^{(G,b,c)})$ as a generalized version of a vertex cover, and an optimal solution would be one of minimum cost (w.r.t. costs b_v). Similarly to the uncapacitated and unweighted case, since the graph is bipartite, ($\mathcal{P}^{(G,b,c)}$) is integral, and thus has an *integral* optimal solution.

$$\begin{aligned} \min \quad & \sum_{v \in V} b_v x_v \\ \text{s.t.} \quad & x_u + x_v \geq c_{uv} \quad \forall (u,v) \in E \\ & x_v \geq 0 \qquad\qquad \forall v \in V \end{aligned} \tag{$\mathcal{P}^{(G,b,c)}$}$$

Claim F.3.4. *Any optimal solution x^* to ($\mathcal{P}^{(G,b,c)}$) has $x^*_v \leq W$ for all $v \in V$.*

Proof. If this is not the case, we can decrease x^*_v without violating any of the constraints, and the objective value $\sum b_v x_v$ becomes smaller. □

This motivates the following feasibility polytope ($\mathcal{P}_F^{(G,b,c)}$), which can be used to check if ($\mathcal{P}^{(G,b,c)}$) has a solution with objective value at most F, for any integer $F \in \mathbb{Z}$.

$$\begin{aligned} &\sum_{v \in V} b_v x_v \leq F + \tfrac{1}{3} \\ &x_u + x_v \geq c_{uv} && \forall (u,v) \in E \\ &0 \leq x_v \leq W + 1 && \forall v \in V \end{aligned} \tag{$\mathcal{P}_F^{(G,b,c)}$}$$

Modifications to the algorithm. Like for the unweighted and uncapacitated case, we can show that if this polytope is non-empty, then it has significantly large volume (Lemma F.3.5, whose proof is in Section F.5). This means that the cutting plane algorithm can terminate whenever the volume becomes too small. The only modifications we need to make to Algorithm F.1 are thus the following:

- We start with a larger initial polytope $P_0 = [0, W+1]^V \cap \{x \in \mathbb{R}^V : \sum b_v x_v \leq F + \frac{1}{3}\}$.
- When we check for (and add) violating constraints we also use the edge-cost c_{uv}. That is an edge (u, v, c_{uv}) is violating if $p_i^u + p_i^v < c_{uv}$, and the corresponding constraint we add is "$x_u + x_v \geq c_{uv}$".
- We terminate when the volume is less than $(\frac{1}{20nW})^n$ (see Lemma F.3.5).

Lemma F.3.5 (Generalization of Lemma F.2.3)**.** *If* ($\mathcal{P}_F^{(G,b,c)}$) *is non-empty, then* $\mathrm{vol}(\mathcal{P}_F^{(G,b,c)}) \geq \left(\frac{1}{20nW}\right)^n$.

Observation F.3.6 (Generalization of Claim F.2.5)**.** *We can communicate a single violated constraint with* $2\log(n) + \log(W) + 1 = O(\log(nW))$ *bits of communication.*

Total communication. The generalized algorithm will start with initial polytope $P_0 \subseteq [0, W+1]^n$, and terminate whenever $\mathrm{vol}(P_i)$ becomes smaller than $\left(\frac{1}{20nW}\right)^n$ (Lemma F.3.5). In each iteration the volume is cut down by a constant fraction (Lemma F.2.8). Hence we need $O\left(\log\left(W^n / \left(\frac{1}{20nW}\right)^n\right)\right) = O(n\log(nW))$ iterations. Each iteration needs $O(\log(nW))$ bits of communication, for a total of $O(n\log^2(nW))$ bits of communication (Observation F.3.6), proving Lemma F.3.3. We also note that the same standard trick to convert the *decision* version to the *optimization* version (Section F.2.2) works here as well.

F.3.3 Application: Unique Bipartite Perfect Matching

In the *unique bipartite matching problem*, or UBPM for short, we are asked to determine if an (unweighted) bipartite graph has a *unique* perfect matching. The interplay between UBPM and BPM is quite subtle, by some measures, e.g. certificate

complexity, the former is known to be strictly harder than the latter, in other settings, such as sequential, simple near linear time algorithms for UBPM have been known since the turn of the century [GKT01]. While for BPM, only a very recent line of work, employing heavy machinery from continuous optimization and dynamic data structures culminated in a near linear time algorithm for BPM [CKLPPS22]. In this section, we show that our upper bounds also hold for the UBPM problem, both for the communication and query models.

Theorem F.3.7. *The* UBPM *problem can be solved in:*

- $O(n \log^2 n)$ *bits of communication in the deterministic two-party edge-partition communication model.*
- $O(n \log^2 n)$ *deterministic OR-queries.*
- $O(n \log^2 n)$ *randomized XOR-queries, w.h.p.*
- $\tilde{O}(n\sqrt{n})$ *quantum edge queries.*

Proof sketch. (Formal proof can in Section F.5).
Our main idea on how to solve UBPM can be summarized in two stages:

(1) First find a perfect matching M (see Theorem F.2.1).

(2) Assign weights to the edges as follows: $c_e = 1$ if $e \in M$, and $c_e = 2$ otherwise. After this, we find a *max-cost* perfect matching M' (see Theorem F.3.1).

If $M' \neq M$, we have proved that the perfect matching is not unique. Conversely, if $M' = M$, then M must be the *unique* perfect matching, since any other perfect matching (if they would exist) has higher cost. In the communication setting, this argument suffices. For the query models, however, one needs to be a bit more careful since we have not defined what, for example, an OR-query means in the *max-cost* setting. The formal proof of this—which is straightforward, although a bit technical—can be found in Section F.5. The other query-models follow from similar reductions as those in Section F.2.3. □

Remark F.3.8. We also note that there are alternative ways to solve UBPM after one has solved BPM. Instead of solving *max-cost matching* as the second step, one can instead determine if an alternating cycle (that is a cycle in G where every other edge is in M) exist. The perfect matching M is *unique* if and only if no such cycle exist. Note that finding such a cycle can easily be done in, for example, $O(n \log n)$ OR-queries by running depth-first-searches to detect a directed cycle in the directed residual graph (edges in M go from $R \to L$, other edges from $L \to R$).

F.4 Communication Lower Bounds

In this section we discuss three communication problems related to OR-queries, AND-queries, and XOR-queries respectively. We show simple lower bounds on the communication complexity of these three problems, and argue that this implies corresponding query lower bounds.

We summarize our obtained query lower bounds below in Theorem F.4.1. Note that our lower bounds are asymptotically the same as those obtained in [BN21]. However, while [BN21] employs rather sophisticated mathematical machinery in order to obtain these lower bounds, we obtain them via simple communication complexity reductions from well known functions. In addition to the already known lower bounds of [BN21], our technique also shows one new result: namely that the $\Omega(n^2)$ AND-query lower bound even holds for **randomized** algorithms.

Theorem F.4.1. *To solve the bipartite perfect matching (*BPM*) problem one needs to use:*

- $\Omega(n \log n)$ OR*-queries (deterministic).*
- $\Omega(n^2)$ AND*-queries (deterministic or randomized).*
- $\Omega(n^2)$ XOR*-queries (deterministic).*

In this section we give an overview of the main ideas—which are all relatively simple—but we postpone the formal proofs to Section F.6.

Problem setup. We consider a two-party communication setting between two players Alice and Bob. Alice is given a graph $G_A = (V, E_A)$ and Bob a graph $G_B = (V, E_B)$ on the same set of vertices $V = L \cup R$ (where $|L| = |R| = n$). They wish to solve BPM on an aggregate of their graphs. We consider three different types of aggregate graphs (and hence get three different communication problems), naturally corresponding to OR / AND / XOR:

- Union graph $G_\cup = (V, E_A \cup E_B)$.
- Intersection graph $G_\cap = (V, E_A \cap E_B)$.
- Symmetric difference graph $G_\oplus = (V, E_A \oplus E_B)$.

It is not difficult to see that any OR / AND / XOR query algorithms can be simulated in an communication protocol for $G_\cup$ / $G_\cap$ / $G_\oplus$ respectively. As an example, to answer an AND-query over $S \subseteq (L \times R)$ in $G_\cap$, Alice and Bob check locally if $S \subseteq E_A$, respectively if $S \subseteq E_B$, and exchange this information with each-other. This gives the following Lemma F.4.2, which we formally prove in Section F.6.

Lemma F.4.2. *If there is a query-algorithm $\mathcal{A}$ solving the bipartite-perfect-matching problems using q many* OR / AND / XOR *queries, then there is a communication protocol which solve the bipartite-perfect-matching problem on $G_\cup$ / $G_\cap$ / $G_\oplus$ respectively, using $2q$ bits of communication. If $\mathcal{A}$ is deterministic, then so is the communication protocol.*

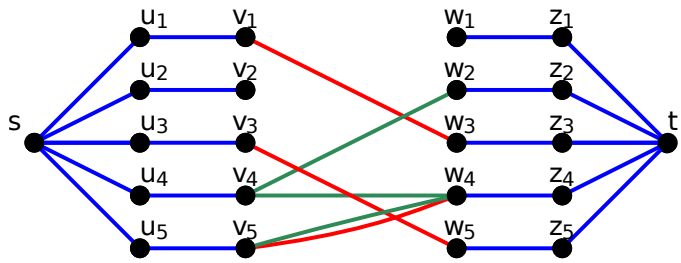


Figure F.1: An example of our graph construction for $G_\cap$. The blue edges are known to both parties to be in the aggregate graph $G_\cap$. Between the v and w layers, Alice owns the red edges and Bob the green. The graph $G_\cap$ has a perfect matching if and only if it has an edge between some v_i and w_j.

Intersection Graph (AND). We construct a difficult instance for solving BPM on $G_\cap$ by reducing from Set-Disjointness on $\Theta(n^2)$ bits. Our construction can be seen in Figure F.1, where determining if $G_\cap$ has a perfect matching boils down to determining if there exists any edge between some vertex v_i and some vertex w_j in $G_\cap$. This is exactly a Set-Disjointness problem of size $\Theta(n^2)$.

Since Set-Disjointness is known to require linear communication in the number of bits [KN97] (both for deterministic and randomized algorithms), we obtain the following Claim F.4.3 whose formal proof can be found in Section F.6.

Claim F.4.3. *Solving* BPM *(or* UBPM*) on $G_\cap$ requires $\Omega(n^2)$ bits of communication, even if public randomness is allowed.*

Symmetric Difference Graph (XOR). Our communication lower bound of $G_\oplus$ is very similar to the lower bound on $G_\cap$. We use the same graph-structure (Figure F.1), but determining if there is any (v_i, w_j) edge in $G_\oplus$ now corresponds to solving the Equality problem (again on $\Theta(n^2)$ bits) instead of the Set-Disjointness problem. This is since an edge (v_i, w_j) exists if and only if exactly one of Alice or Bob have it, which is if and only if the set of Alice's edges does not equal the set of Bob's edges.

It is well known that Equality exhibits a large gap between its deterministic and randomized communication complexity. While the former is known to be $\Omega(n)$, the latter requires only $O(1)$ bits of communication in the presence of shared randomness (with error probability $< \frac{1}{3}$) [KN97]. This might also provide some intuition why we, in the case of XOR-queries, obtain the separation of $\tilde{O}(n)$ randomized upper bound

(Claim F.2.12) vs $\Omega(n^2)$ deterministic lower bound. The full proof of Claim F.4.4 can be found in Section F.6.

Claim F.4.4. *Solving* BPM *on* $G_{\oplus}$ *requires* $\Omega(n^2)$ *bits of communication for any deterministic protocol.*

Union Graph (OR). Our lower bound for $G_{\cup}$ follows a similar vein, but the graph construction is a bit different. By a standard reduction, the (s,t)-reachability problem on a (not-necessarily-bipartite) n-vertex graph can be solved by solving bipartite perfect-matching on a graph on $2n$ vertices. The (s,t)-reachability problem, in the edge-partition setting, is known to need $\Omega(n \log n)$ bits of communication for any deterministic protocol [HMT88]. Proving any super-linear *randomized* lower bound still remains open. The full proof of Claim F.4.5 can be found in Section F.6.

Claim F.4.5. *Solving* BPM *on* $G_{\cup}$ *requires* $\Omega(n \log n)$ *bits of communication for any deterministic protocol.*

Acknowledgement

Jan van den Brand is partially funded by ONR BRC grant N00014-18-1-2562 and by the Simons Institute for the Theory of Computing through a Simons-Berkeley Postdoctoral Fellowship.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. Nanongkai was also partially supported by the Swedish Research Council (Reg. No. 2019-05622).

The authors would like to thank Gal Beniamini for fruitful discussions and anonymous reviewers of FOCS 2022 for their useful suggestions.

APPENDIX

F.5 Omitted proofs from Section F.3.2

Lemma F.3.5 (Generalization of Lemma F.2.3)**.** *If* $(\mathcal{P}_F^{(G,b,c)})$ *is non-empty, then* $\text{vol}(\mathcal{P}_F^{(G,b,c)}) \geq \left(\frac{1}{20nW}\right)^n$.

Proof. Similarly to Lemma F.2.3, let x be some integral feasible solution to $(\mathcal{P}_F^{(G,b,c)})$, we know such a solution exists since $(\mathcal{P}^{(G,b,c)})$ has an optimal integer solution, and by assumption that $(\mathcal{P}_F^{(G,b,c)})$ is not empty, we can deduce that its value is at most $F + \frac{1}{3}$, which is at most F since it is integer. For the same reason, we can also, by Claim F.3.4, assume w.l.o.g. that $x_v \leq W$ for all $v \in V$. Our goal is to prove that $(\mathcal{P}_F^{(G,b,c)})$ contains a cube of dimensions $\frac{1}{20nW}$, thus concluding the proof. Note that for each $v \in V$ we can independently increase the value of x_v by any value

in the range $[0, \frac{1}{20nW}]$ while maintaining the feasibility of the resulting point w.r.t. $(\mathcal{P}_F^{(G,b,c)})$. This is due to the following observations.

- The constraint $\sum_{v \in V} b_v x_v \leq F + \frac{1}{3}$ remains valid as as x has value F, and we increase each coordinate by at most $\frac{1}{20nW}$, as there are $2n$ vertices and $b_v \leq W$ for all $v \in V$, it means that we increase the value of the solution by at most $\frac{1}{10}$, which is less than $\frac{1}{3}$.
- All edge constrains $x_v + x_u \geq c_{uv}$ for all $(u, v) \in E$ clearly remain valid as we are only increasing the value of variables.
- As F.3.4 allows us to assume that $x_v \leq W$ for all $v \in V$ for the original x, the constraints $0 \leq x_v \leq W + 1$ for all $v \in V$ remain valid as well as each entry is increased by at most $\left(\frac{1}{20nW}\right)$. □

Theorem F.3.7. *The* UBPM *problem can be solved in:*

- $O(n \log^2 n)$ *bits of communication in the deterministic two-party edge-partition communication model.*
- $O(n \log^2 n)$ *deterministic OR-queries.*
- $O(n \log^2 n)$ *randomized XOR-queries, w.h.p.*
- $\tilde{O}(n\sqrt{n})$ *quantum edge queries.*

Proof. Here we give the formal proof that we can determine if an unweighted bipartite graph $G = (V, E)$ (with $V = L \cup R$, $|L| = |R| = n$) has a *unique* bipartite matching in all our different models. We show that $O(n \log^2 n)$ OR-queries suffices, and the other models follows similarly as in Section F.2.3.

We start by obtaining a maximum matching M of G, by Lemma F.2.4. If M is not a perfect matching, G admits no perfect matching, and we are done. From now on assume that M is a perfect matching.

We now construct an instance of a weighted matching. Let $c_e = 10n$ if $e \in M$ and $c_e = 10n + 1$ otherwise. We wish to solve the max-cost matching problem with edge costs c. We will soon explain how to do this with OR-queries, but first we show how doing this completes the proof.

Let M' be a max-cost matching in G with respect to the costs c. Note that M has cost $10n^2$, so we know that M' has cost at least $10n^2$ too. First we argue that M' must be a perfect matching of G. This is because any non-perfect matching has weight at most $(n-1)(10n+1) < 10n^2$.

- If $M' \neq M$, we have found two perfect matchings of the graph, and are done (G does not have a *unique* perfect matching).

- If $M' = M$, we conclude that M is the *unique* perfect matching in the graph. This is since any other perfect matching of G, if they would exist, would have had strictly larger cost.

Now we return to explaining how to solve the max-cost (w.r.t. the cost c) matching using only OR-queries. Note that we already know the weights of all (potential) edges (either $10n$ if they are in M, which we know; or $10n+1$ otherwise), and this is not something which needs to be found out using OR-queries. By the above discussion, it suffices to check if any matching of weight strictly more than $10n^2$ exist or not. Hence we use a version of ($\mathcal{P}_F^{(G,b,c)}$) with c being our costs, b the all-ones vector, and $F = 10n^2$, to obtain the following polytope ($\mathcal{P}^{\mathsf{UBPM}}$):

$$\begin{aligned} &\sum_{v \in V} x_v \leq 10n^2 + \tfrac{1}{3} \\ &x_u + x_v \geq 10n && \forall (u,v) \in M \\ &x_u + x_v \geq 10n + 1 && \forall (u,v) \in E \setminus M \\ &0 \leq x_v \leq 10n + 2 && \forall v \in V \end{aligned} \tag{$\mathcal{P}^{\mathsf{UBPM}}$}$$

If this polytope is feasible, then M must be the unique perfect matching, and if it is feasible, another perfect matching M' of higher cost exist.

We note that in the above polytope (which is a special case of the ($\mathcal{P}_F^{(G,b,c)}$) polytope from Section F.3), we know all the constraints initially except the "$x_u + x_v \geq 10n$ for $(u,v) \in E \setminus M$" constraints. Hence, when running the cutting planes algorithm we may start with all other constraints in our initial polytope. Now, to implement the FindViolatingEdge-subroutine (a.k.a. the separation oracle) given a point p_i, we binary search (with OR-queries) over the set of potentially violated edges, i.e. the set $S = \{(u,v) \in L \times R \mid p_i^u + p_i^v < 10n + 1, (u,v) \notin M\}$ (like in Claim F.2.5, but now this set S is a bit different for our problem). Otherwise, the cutting planes algorithm is exactly the same as for the max-cost b-matching in Section F.3.2. □

F.6 Omitted proofs from Section F.4

Lemma F.4.2. *If there is a query-algorithm $\mathcal{A}$ solving the bipartite-perfect-matching problems using q many* OR / AND / XOR *queries, then there is a communication protocol which solve the bipartite-perfect-matching problem on $G_\cup$ / $G_\cap$ / $G_\oplus$ respectively, using $2q$ bits of communication. If $\mathcal{A}$ is deterministic, then so is the communication protocol.*

Proof. Alice and Bob can simulate a query on the aggregate graph with a single bit sent in each direction as follows:

- An OR-query $S \subseteq L \times R$ can be simulated on $G_\cup$: Alice and Bob locally check if $|S \cap E_A| > 0$, respectively $|S \cap E_B| > 0$, and communicates this to the other party.

- An AND-query $S \subseteq L \times R$ can be simulated on $G_\cap$: Alice and Bob locally check if $|S \cap E_A| = |S|$, respectively $|S \cap E_B| = |S|$, and communicates this to the other party.
- An XOR-query $S \subseteq L \times R$ can be simulated on $G_\oplus$: Alice and Bob locally compute the parity of $|S \cap E_A|$, respectively $|S \cap E_B|$, and communicates this to the other party. If they have the same parity, then the parity of $|S \cap (E_B \oplus E_A)|$ is even, otherwise it is odd. □

Claim F.4.3. *Solving* BPM *(or* UBPM*) on $G_\cap$ requires $\Omega(n^2)$ bits of communication, even if public randomness is allowed.*

Proof. Let k be an integer and suppose Alice and Bob are tasked to solve a Set-Disjointness problem on k^2 bits. That is, suppose Alice is given a subset $A \subseteq [k] \times [k]$ and Bob is given a subset $B \subseteq [k] \times [k]$, and they want to determine if $A \cap B$ is empty or not. This is known to require $\Omega(k^2)$ bits of communication, even if public randomness is allowed [Raz92].

We now proceed by setting up two bipartite graphs $G_A = (L \cup R, E_A)$ and $G_B = (L \cup R, E_B)$ such that the graph $G_\cap = (L \cup R, E_A \cap E_B)$ has a perfect matching if and only if $A \cap B \neq \emptyset$. For an illustration, see Figure F.1.

The graph will have $4k + 2$ vertices. Let $L = \{s,\ v_1, v_2, \ldots, v_k,\ z_1, z_2, \ldots, z_k\}$ and $R = \{t,\ u_1, u_2, \ldots, u_k,\ w_1, w_2, \ldots, w_k\}$. The edges (s, u_i), (u_i, v_i), (w_i, z_i) and (z_i, t) (for all $i \in [k]$) will be in both Alice's and Bob's graphs. Note that the edges (u_i, v_i) and (w_i, z_i) form an almost-perfect-matching in $G_\cap$: all vertices except s and t are matched. Hence, a perfect-matching exists in $G_\cap$ if and only if there is an augmenting path (with respect to the almost-perfect-matching) between s and t in the graph.

Additionally, for every $(i, j) \in A$ we add the edge (v_i, w_j) to Alice's edges, and similarly for every edge $(i, j) \in B$ we add the edge (v_i, w_j) to Bob's edges.

- If $(i, j) \in A \cap B$ exists, then (v_i, w_j) is an edge of $G_\cap$, and $G_\cap$ has a perfect matching: $(s, u_i, v_i, w_j, z_j, t)$ forms the desired (s, t)-augmenting path.
- On the other hand, if $A \cap B = \emptyset$, then $G_\cap$ has no perfect matching since s and t are in different components in the graph and hence no augmenting path between them exists.

Note that the version of Set-Disjointness where we are promised that $|A \cap B| \leq 1$ is still difficult (i.e. also has an $\Omega(n^2)$ communication lower bound) [Raz92]. This means that our lower bound also holds for the *unique* bipartite matching problem (UBPM), as there can never be more than one perfect matching in this promise version. □

Claim F.4.4. *Solving* BPM *on $G_\oplus$ requires $\Omega(n^2)$ bits of communication for any deterministic protocol.*

Proof. Let k be an integer and suppose Alice and Bob are tasked to solve a Equality problem on k^2 bits. That is, suppose Alice is given a subset $A \subseteq [k] \times [k]$ and Bob is given a subset $B \subseteq [k] \times [k]$, and they want to determine if $A = B$ (this is equivalent to determine if $A \oplus B = \emptyset$). This is known to require $\Omega(k^2)$ bits of communication for any deterministic algorithm.

We use a similar graph construction on $4k+2$ vertices as in the $G_\cap$-query setting (again, for an illustration, see Figure F.1). The edges (s, u_i), (u_i, v_i), (w_i, z_i) and (z_i, t) (for all $i \in [k]$) will be in be in Alice's graph, but not Bob's (so that all these edges will be $G_\oplus$). Again, note that a perfect-matching exists in $G_\oplus$ if and only if there is an augmenting path (with respect to the almost-perfect-matching $\{(u_i, v_i), (w_i, z_i) : i \in [k]\}$) between s and t in the graph.

Additionally, we add the edge (v_i, w_j) to Alice's graph if $(i, j) \in A$, and similarly add the edge (v_i, w_j) to Bob's graph if $(i, j) \in B$. That is the edge (v_i, w_j) is in $G_\oplus$ if and only if $(i, j) \in A \oplus B$.

- If $(i, j) \in A \oplus B$ exist, then then (v_i, w_j) is an edge of $G_\oplus$, and $G_\oplus$ has a perfect matching: $(s, u_i, v_i, w_j, z_j, t)$ forms the desired (s, t)-augmenting path.
- On the other hand, if $A \oplus B = \emptyset$, then $G_\oplus$ has no perfect matching since s and t are in different components in the graph and hence no augmenting path between them exists. □

Claim F.4.5. *Solving* BPM *on* $G_\cup$ *requires* $\Omega(n \log n)$ *bits of communication for any deterministic protocol.*

Proof. Suppose the two parties are given an instance of (s, t)-connectivity. That is Alice is given edges F_A and Bob F_B in a k-vertex (not-necessarily-bipartite) graph $H = (V, F_A \cup F_B)$. They are also both given two vertices $s, t \in V$ and want to determine if s and t are in the same connected component in H. This is known to require $\Omega(n \log n)$ bits of communication.

We proceed by constructing bipartite graphs $G_A = (L \cup R, E_A)$ and $G_B = (L \cup R, E_B)$ for Alice and Bob, such that $G_\cup = (L \cup R, E_A \cup E_B)$ has a perfect matching if and only if s and t are connected in H.

We let $L = \{v : v \in V \setminus \{s\}\}$ and $R = \{v' : v \in V \setminus \{t\}\}$. Note that $|L| = |R| = k - 1$. Let the edge (v, v') be in both E_A and E_B (and thus also an edge of $G_\cup$) for all $v \in V \setminus \{s, t\}$. These edges form an almost-perfect matching of size $k - 2$: all vertices except s and t' are matched. Again, $G_\cup$ has a perfect matching if and only if there is an augmenting path (with respect to this partial almost-perfect matching) between s and t in the graph.

For each edge $(v, u) \in F_A$, we add (v, u') (unless $v = t$) and (v', u) (unless $v = s$) to Alice's edges E_A. We do the same for Bob.

Now there is an (s, t)-path in H if and only if there is an augmenting path (w.r.t. the almost-perfect matching $\{(v, v') : v \in V \setminus \{s, t\}\}$) between s and t in $G_\cup$. Indeed, a path $(s, v_1, v_2, \ldots, v_r, t)$ in H corresponds to the augmenting path $(s, v'_1, v_1, v'_2, v_2, \ldots, v'_r, v_r, t)$ in $G_\cup$. □

Bibliography

[AA05] Noga Alon and Vera Asodi. “Learning a Hidden Subgraph”. In: *SIAM J. Discret. Math.* 18.4 (2005), pp. 697–712 (cit. on pp. 338, 349).

[AB19a] Hasan Abasi and Nader H. Bshouty. “On Learning Graphs with Edge-Detecting Queries”. In: *ALT*. Vol. 98. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3–30 (cit. on pp. 338, 349).

[AB19b] Sepehr Assadi and Aaron Bernstein. “Towards a Unified Theory of Sparsification for Matching Problems”. In: *SOSA*. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 11:1–11:20 (cit. on p. 335).

[AB21] Sepehr Assadi and Soheil Behnezhad. “On the Robust Communication Complexity of Bipartite Matching”. In: *APPROX-RANDOM*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 48:1–48:17 (cit. on p. 335).

[ABKRS04] Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. “Learning a Hidden Matching”. In: *SIAM J. Comput.* 33.2 (2004), pp. 487–501 (cit. on pp. 338, 349).

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. “Polynomial pass lower bounds for graph streaming algorithms”. In: *STOC*. ACM, 2019, pp. 265–276 (cit. on p. 342).

[ACK21] Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. “Graph Connectivity and Single Element Recovery via Linear and OR Queries”. In: *ESA*. Vol. 204. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 7:1–7:19 (cit. on p. 337).

[AD21] Sepehr Assadi and Aditi Dudeja. “A Simple Semi-Streaming Algorithm for Global Minimum Cuts”. In: *SOSA*. SIAM, 2021, pp. 172–180 (cit. on p. 341).

[AJJST22] Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. “Semi-Streaming Bipartite Matching in Fewer Passes and Optimal Space”. In: *SODA*. SIAM, 2022, pp. 627–669 (cit. on p. 341).

[AK20] Mohamad Ahmadi and Fabian Kuhn. “Distributed Maximum Matching Verification in CONGEST”. In: *DISC*. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 37:1–37:18 (cit. on pp. 335, 341).

[AKL17] Sepehr Assadi, Sanjeev Khanna, and Yang Li. “On Estimating Maximum Matching Size in Graph Streams”. In: *SODA*. SIAM, 2017, pp. 1723–1742 (cit. on p. 335).

[AKLY16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. “Maximum Matchings in Dynamic Graph Streams and the Simultaneous Communication Model”. In: *SODA*. SIAM, 2016, pp. 1345–1364 (cit. on p. 335).

[AKO18] Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. "Distributed Approximate Maximum Matching in the CONGEST Model". In: *DISC*. Vol. 121. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 6:1–6:17 (cit. on p. 341).

[AL21] Arinta Auza and Troy Lee. "On the query complexity of connectivity with global queries". In: *CoRR* abs/2109.02115 (2021) (cit. on pp. 338, 349).

[ALT21] Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. "An Auction Algorithm for Bipartite Matching in Streaming and Massively Parallel Computation Models". In: *SOSA*. SIAM, 2021, pp. 165–171 (cit. on p. 335).

[Amb02] Andris Ambainis. "Quantum Lower Bounds by Quantum Arguments". In: *J. Comput. Syst. Sci.* 64.4 (2002), pp. 750–767 (cit. on p. 337).

[AMV20] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. "Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs". In: *FOCS*. IEEE, 2020, pp. 93–104 (cit. on p. 338).

[AR20a] Udit Agarwal and Vijaya Ramachandran. "Faster Deterministic All Pairs Shortest Paths in Congest Model". In: *SPAA*. ACM, 2020, pp. 11–21 (cit. on p. 341).

[AR20b] Sepehr Assadi and Ran Raz. "Near-Quadratic Lower Bounds for Two-Pass Graph Streaming Algorithms". In: *FOCS*. IEEE, 2020, pp. 342–353 (cit. on pp. 335, 336, 341).

[AV20] Nima Anari and Vijay V. Vazirani. "Matching Is as Easy as the Decision Problem, in the NC Model". In: *ITCS*. Vol. 151. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 54:1–54:25 (cit. on p. 335).

[Ben22a] Gal Beniamini. "Algebraic Representations of Unique Bipartite Perfect Matching". In: *CoRR* abs/2203.01071 (2022) (cit. on pp. 335, 337, 338).

[Ben22b] Gal Beniamini. "The Approximate Degree of Bipartite Perfect Matching". In: *CoRR* abs/2004:14318 (2022) (cit. on pp. 335–337, 340).

[Ber09] Dimitri P. Bertsekas. "Auction Algorithms". In: *Encyclopedia of Optimization*. Springer, 2009, pp. 128–132 (cit. on p. 335).

[BFS86] László Babai, Peter Frankl, and Janos Simon. "Complexity classes in communication complexity theory (preliminary version)". In: *FOCS*. IEEE Computer Society, 1986, pp. 337–347 (cit. on p. 335).

[BHR19] Aaron Bernstein, Jacob Holm, and Eva Rotenberg. "Online Bipartite Matching with Amortized $O(\log^2 n)$ Replacements". In: *J. ACM* 66.5 (2019), 37:1–37:23 (cit. on p. 335).

[BHRRS18] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. "Edge Estimation with Independent Set Oracles". In: *ITCS*. Vol. 94. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 38:1–38:21 (cit. on pp. 338, 349).

[BLLSSSW21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances". In: *STOC*. ACM, 2021, pp. 859–869 (cit. on p. 338).

[BLNPSSSW20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs". In: *FOCS*. IEEE, 2020, pp. 919–930 (cit. on pp. 335, 336, 338, 351, 352).

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. "Solving tall dense linear programs in nearly linear time". In: *STOC*. ACM, 2020, pp. 775–788 (cit. on p. 338).

[BN19] Aaron Bernstein and Danupon Nanongkai. "Distributed exact weighted all-pairs shortest paths in near-linear time". In: *STOC*. ACM, 2019, pp. 334–342 (cit. on p. 341).

[BN21] Gal Beniamini and Noam Nisan. "Bipartite perfect matching as a real polynomial". In: *STOC*. ACM, 2021, pp. 1118–1131 (cit. on pp. 335–337, 340, 348, 356).

[Bra20] Jan van den Brand. "A Deterministic Linear Program Solver in Current Matrix Multiplication Time". In: *SODA*. SIAM, 2020, pp. 259–278 (cit. on p. 338).

[BW02] Harry Buhrman and Ronald de Wolf. "Complexity measures and decision tree complexity: a survey". In: *Theor. Comput. Sci.* 288.1 (2002), pp. 21–43 (cit. on p. 351).

[CGLNPS20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. "A Deterministic Algorithm for Balanced Cut with Applications to Dynamic Connectivity, Flows, and Beyond". In: *FOCS*. IEEE, 2020, pp. 1158–1167 (cit. on p. 342).

[CK07] Amit Chakrabarti and Subhash Khot. "Improved lower bounds on the randomized complexity of graph properties". In: *Random Struct. Algorithms* 30.3 (2007), pp. 427–440 (cit. on p. 337).

[CKLM18] Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. "Simulation beats richness: new data-structure lower bounds". In: *STOC*. ACM, 2018, pp. 1013–1020 (cit. on p. 337).

[CKLPPS22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum flow and minimum-cost flow in almost-linear time". In: *CoRR* abs/2203.00671 (2022) (cit. on pp. 335, 338, 355).

[CKPSSY21] Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. "Almost optimal super-constant-pass streaming lower bounds for reachability". In: *STOC*. ACM, 2021, pp. 570–583 (cit. on pp. 335, 336, 341).

[CLS19] Michael B. Cohen, Yin Tat Lee, and Zhao Song. "Solving linear programs in the current matrix multiplication time". In: *STOC*. ACM, 2019, pp. 938–942 (cit. on p. 338).

[CM20] Shiri Chechik and Doron Mukhtar. "Single-Source Shortest Paths in the CONGEST Model with Improved Bound". In: *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*. ACM, 2020, pp. 464–473 (cit. on p. 341).

[CMSV17] Michael B Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. "Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $O(m^{10/7} \log W)$ Time". In: *SODA*. SIAM. 2017, pp. 752–771 (cit. on p. 338).

[CQ21] Chandra Chekuri and Kent Quanrud. "Isolating Cuts, (Bi-)Submodularity, and Faster Algorithms for Connectivity". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 50:1–50:20 (cit. on p. 342).

[DEMN21] Michal Dory, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Distributed weighted min-cut in nearly-optimal time". In: *STOC*. ACM, 2021, pp. 1144–1153 (cit. on p. 341).

[DHHM06] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. "Quantum Query Complexity of Some Graph Problems". In: *SIAM J. Comput.* 35.6 (2006), pp. 1310–1328 (cit. on p. 337).

[DHKKNPPW12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. "Distributed Verification and Hardness of Distributed Approximation". In: *SIAM J. Comput.* 41.5 (2012), pp. 1235–1265 (cit. on p. 341).

[DNO19] Shahar Dobzinski, Noam Nisan, and Sigal Oren. "Economic efficiency requires interaction". In: *Games Econ. Behav.* 118 (2019), pp. 589–608 (cit. on pp. 335, 336).

[DP89] Pavol Duris and Pavel Pudlák. "On the Communication Complexity of Planarity". In: *FCT*. Vol. 380. Lecture Notes in Computer Science. Springer, 1989, pp. 145–147 (cit. on p. 335).

[DS08] Samuel I. Daitch and Daniel A. Spielman. "Faster approximate lossy generalized flow via interior point algorithms". In: *STOC*. ACM, 2008, pp. 451–460 (cit. on p. 338).

[Elk20] Michael Elkin. "Distributed Exact Shortest Paths in Sublinear Time". In: *J. ACM* 67.3 (2020), 15:1–15:36 (cit. on p. 341).

[FF56] Lester Randolph Ford and Delbert R Fulkerson. "Maximal flow through a network". In: *Canadian journal of Mathematics* 8 (1956), pp. 399–404 (cit. on p. 335).

[FGLPSY21] Sebastian Forster, Gramoz Goranci, Yang P. Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. "Minor Sparsifiers and the Distributed Laplacian Paradigm". In: *FOCS*. IEEE, 2021, pp. 989–999 (cit. on pp. 335, 341).

[FGT21] Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. “Bipartite Perfect Matching is in Quasi-NC”. In: *SIAM J. Comput.* 50.3 (2021) (cit. on p. 335).

[FKMSZ05] 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 (cit. on pp. 335, 336, 341).

[FN18] Sebastian Forster and Danupon Nanongkai. “A Faster Distributed Single-Source Shortest Paths Algorithm”. In: *FOCS*. IEEE Computer Society, 2018, pp. 686–697 (cit. on p. 341).

[Gal16] François Le Gall. “Further Algebraic Algorithms in the Congested Clique Model and Applications to Graph-Theoretic Problems”. In: *DISC*. Vol. 9888. Lecture Notes in Computer Science. Springer, 2016, pp. 57–70 (cit. on p. 341).

[GG17] Shafi Goldwasser and Ofer Grossman. “Bipartite Perfect Matching in Pseudo-Deterministic NC”. In: *ICALP*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 87:1–87:13 (cit. on p. 335).

[GHS83] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. “A Distributed Algorithm for Minimum-Weight Spanning Trees”. In: *ACM Trans. Program. Lang. Syst.* 5.1 (1983), pp. 66–77 (cit. on p. 341).

[GKK12] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. “On the communication and streaming complexity of maximum bipartite matching”. In: *SODA*. SIAM, 2012, pp. 468–485 (cit. on pp. 335, 336).

[GKKLP18] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. “Near-Optimal Distributed Maximum Flow”. In: *SIAM J. Comput.* 47.6 (2018), pp. 2078–2117 (cit. on p. 341).

[GKT01] Harold N. Gabow, Haim Kaplan, and Robert Endre Tarjan. “Unique Maximum Matching Algorithms”. In: *J. Algorithms* 40.2 (2001), pp. 159–183 (cit. on pp. 338, 355).

[GL18] Mohsen Ghaffari and Jason Li. “Improved distributed algorithms for exact shortest paths”. In: *STOC*. ACM, 2018, pp. 431–444 (cit. on p. 341).

[GNT20] Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. “Faster Algorithms for Edge Connectivity via Random 2-Out Contractions”. In: *SODA*. SIAM, 2020, pp. 1260–1279 (cit. on p. 341).

[GO16] Venkatesan Guruswami and Krzysztof Onak. “Superlinear Lower Bounds for Multipass Graph Processing”. In: *Algorithmica* 76.3 (2016), pp. 654–683 (cit. on pp. 335, 341).

[Gro96] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *STOC*. ACM, 1996, pp. 212–219 (cit. on p. 351).

[Grü60] Branko Grünbaum. "Partitions of mass-distributions and of convex bodies by hyperplanes." In: *Pacific Journal of Mathematics* 10 (1960), pp. 1257–1261 (cit. on p. 346).

[Haj91] Péter Hajnal. "An Omega($n^{4/3}$) lower bound on the randomized complexity of graph properties". In: *Comb.* 11.2 (1991), pp. 131–143 (cit. on p. 337).

[HHL18] Hamed Hatami, Kaave Hosseini, and Shachar Lovett. "Structure of Protocols for XOR Functions". In: *SIAM J. Comput.* 47.1 (2018), pp. 208–217 (cit. on p. 337).

[HK73] 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 (cit. on p. 335).

[HKN21] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "A Deterministic Almost-Tight Distributed Algorithm for Approximating Single-Source Shortest Paths". In: *SIAM J. Comput.* 50.3 (2021) (cit. on p. 341).

[HMT06] Thanh Minh Hoang, Meena Mahajan, and Thomas Thierauf. "On the Bipartite Unique Perfect Matching Problem". In: *ICALP*. Vol. 4051. Lecture Notes in Computer Science. Springer, 2006, pp. 453–464 (cit. on p. 338).

[HMT88] András Hajnal, Wolfgang Maass, and György Turán. "On the Communication Complexity of Graph Properties". In: *STOC*. ACM, 1988, pp. 186–191 (cit. on pp. 335, 358).

[HRVZ15] Zengfeng Huang, Bozidar Radunovic, Milan Vojnovic, and Qin Zhang. "Communication Complexity of Approximate Matching in Distributed Graphs". In: *STACS*. Vol. 30. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 460–473 (cit. on p. 335).

[HRVZ20] Zengfeng Huang, Bozidar Radunovic, Milan Vojnovic, and Qin Zhang. "Communication complexity of approximate maximum matching in the message-passing model". In: *Distributed Comput.* 33.6 (2020), pp. 515–531 (cit. on p. 335).

[HWZ21] Bernhard Haeupler, David Wajc, and Goran Zuzic. "Universally-optimal distributed algorithms for known topologies". In: *STOC*. ACM, 2021, pp. 1166–1179 (cit. on p. 341).

[II86] Amos Israeli and Alon Itai. "A Fast and Simple Randomized Parallel Algorithm for Maximal Matching". In: *Inf. Process. Lett.* 22.2 (1986), pp. 77–80 (cit. on p. 341).

[IKLSW12] Gábor Ivanyos, Hartmut Klauck, Troy Lee, Miklos Santha, and Ronald de Wolf. "New bounds on the classical and quantum communication complexity of some graph properties". In: *FSTTCS*. Vol. 18. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 148–159 (cit. on pp. 335, 336).

[JST20] Yujia Jin, Aaron Sidford, and Kevin Tian. "Semi-Streaming Bipartite Matching in Fewer Passes and Less Space". In: *CoRR* abs/2011.03495 (2020) (cit. on p. 335).

[Kap21] Michael Kapralov. "Space Lower Bounds for Approximating Maximum Matching in the Edge Arrival Model". In: *SODA*. SIAM, 2021, pp. 1874–1893 (cit. on p. 335).

[KKS14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. "Approximating matching size from random streams". In: *SODA*. SIAM, 2014, pp. 734–751 (cit. on p. 335).

[KM93] Eyal Kushilevitz and Yishay Mansour. "Learning Decision Trees Using the Fourier Spectrum". In: *SIAM J. Comput.* 22.6 (1993), pp. 1331–1348 (cit. on p. 337).

[KMNT20] Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. "Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples". In: *SODA*. SIAM, 2020, pp. 1753–1772 (cit. on p. 335).

[KMT21] Michael Kapralov, Gilbert Maystre, and Jakab Tardos. "Communication Efficient Coresets for Maximum Matching". In: *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*. SIAM, 2021, pp. 156–164 (cit. on p. 335).

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. ISBN: 978-0-521-56067-2 (cit. on p. 357).

[KP98] Shay Kutten and David Peleg. "Fast Distributed Construction of Small k-Dominating Sets and Applications". In: *J. Algorithms* 28.1 (1998), pp. 40–66 (cit. on p. 341).

[KSS84] Jeff Kahn, Michael E. Saks, and Dean Sturtevant. "A topological approach to evasiveness". In: *Comb.* 4.4 (1984), pp. 297–306 (cit. on p. 337).

[KUW85] Richard M. Karp, Eli Upfal, and Avi Wigderson. "Constructing a Perfect Matching is in Random NC". In: *STOC*. ACM, 1985, pp. 22–32 (cit. on p. 335).

[KVKV11] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Vol. 1. Springer, 2011 (cit. on p. 343).

[KVV85] Dexter Kozen, Umesh V. Vazirani, and Vijay V. Vazirani. "NC Algorithms for Comparability Graphs, Interval Gaphs, and Testing for Unique Perfect Matching". In: *FSTTCS*. Vol. 206. Lecture Notes in Computer Science. Springer, 1985, pp. 496–503 (cit. on p. 338).

[KVV90] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. "An Optimal Algorithm for On-line Bipartite Matching". In: *STOC*. ACM, 1990, pp. 352–358 (cit. on p. 335).

[Lev65] A. Y. Levin. "On an algorithm for the minimization of convex functions over convex functions". In: *Soviet Mathematics Doklady* 160 (1965), pp. 1244–1247 (cit. on pp. 339, 345).

[LL15] Cedric Yen-Yu Lin and Han-Hsuan Lin. "Upper Bounds on Quantum Query Complexity Inspired by the Elitzur-Vaidman Bomb Tester". In: *Computational Complexity Conference*. Vol. 33. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 537–566 (cit. on pp. 336, 337).

[LNPSY21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. "Vertex connectivity in poly-logarithmic max-flows". In: *STOC*. ACM, 2021, pp. 317–329 (cit. on p. 342).

[Lov79] László Lovász. "On determinants, matchings, and random algorithms". In: *FCT*. Akademie-Verlag, Berlin, 1979, pp. 565–574 (cit. on p. 335).

[LP20] Jason Li and Debmalya Panigrahi. "Deterministic Min-cut in Poly-logarithmic Max-flows". In: *FOCS*. IEEE, 2020, pp. 85–92 (cit. on p. 342).

[LP21] Jason Li and Debmalya Panigrahi. "Approximate Gomory-Hu tree is faster than n - 1 max-flows". In: *STOC*. ACM, 2021, pp. 1738–1748 (cit. on p. 342).

[LS14] Yin Tat Lee and Aaron Sidford. "Path Finding Methods for Linear Programming: Solving Linear Programs in Õ(sqrt(rank)) Iterations and Faster Algorithms for Maximum Flow". In: *FOCS*. 2014, pp. 424–433. DOI: 10.1109/FOCS.2014.52 (cit. on p. 338).

[LS20] Yang P. Liu and Aaron Sidford. "Faster energy maximization for faster maximum flow". In: *STOC*. ACM, 2020, pp. 803–814 (cit. on p. 338).

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. "A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization". In: *FOCS*. IEEE Computer Society, 2015, pp. 1049–1065 (cit. on p. 339).

[LSZ20] S. Cliff Liu, Zhao Song, and Hengjie Zhang. "Breaking the n-Pass Barrier: A Streaming Algorithm for Maximum Weight Bipartite Matching". In: *CoRR* abs/2009.06106 (2020) (cit. on p. 341).

[Lub86] Michael Luby. "A Simple Parallel Algorithm for the Maximal Independent Set Problem". In: *SIAM J. Comput.* 15.4 (1986), pp. 1036–1053 (cit. on p. 341).

[Mad13] Aleksander Madry. "Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back". In: *FOCS*. IEEE Computer Society, 2013, pp. 253–262 (cit. on pp. 335, 338).

[Mad16] Aleksander Madry. "Computing maximum flow with augmenting electrical flows". In: *FOCS*. IEEE. 2016, pp. 593–602 (cit. on pp. 335, 338).

[MN20] Sagnik Mukhopadhyay and Danupon Nanongkai. "Weighted min-cut: sequential, cut-query, and streaming algorithms". In: *STOC*. ACM, 2020, pp. 496–509 (cit. on p. 341).

[MN21] Sagnik Mukhopadhyay and Danupon Nanongkai. “A Note on Isolating Cut Lemma for Submodular Function Minimization”. In: *CoRR* abs/2103.15724 (2021) (cit. on p. 342).

[MO09] Ashley Montanaro and Tobias Osborne. “On the communication complexity of XOR functions”. In: *CoRR* abs/0909.3392 (2009) (cit. on p. 337).

[MS04] Marcin Mucha and Piotr Sankowski. “Maximum Matchings via Gaussian Elimination”. In: *FOCS*. IEEE Computer Society, 2004, pp. 248–255 (cit. on p. 335).

[MS20] Nikhil S. Mande and Swagato Sanyal. “On Parity Decision Trees for Fourier-Sparse Boolean Functions”. In: *FSTTCS*. Vol. 182. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 29:1–29:16 (cit. on p. 337).

[Nan14] Danupon Nanongkai. “Distributed approximation algorithms for weighted shortest paths”. In: *STOC*. ACM, 2014, pp. 565–573 (cit. on p. 341).

[NC16] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016 (cit. on p. 351).

[New65] Donald J Newman. “Location of the maximum on unimodal surfaces”. In: *Journal of the ACM (JACM)* 12.3 (1965), pp. 395–398 (cit. on pp. 339, 345).

[Nis21] Noam Nisan. “The Demand Query Model for Bipartite Matching”. In: *SODA*. SIAM, 2021, pp. 592–599 (cit. on pp. 335–337, 340).

[NS14] Danupon Nanongkai and Hsin-Hao Su. “Almost-Tight Distributed Minimum Cut Algorithms”. In: *DISC*. Vol. 8784. Lecture Notes in Computer Science. Springer, 2014, pp. 439–453 (cit. on p. 341).

[PS82] Christos H. Papadimitriou and Michael Sipser. “Communication Complexity”. In: *STOC*. ACM, 1982, pp. 196–200 (cit. on p. 335).

[Raz92] Alexander A. Razborov. “On the Distributional Complexity of Disjointness”. In: *Theor. Comput. Sci.* 106.2 (1992), pp. 385–390 (cit. on pp. 335, 361).

[Ros73] Arnold L. Rosenberg. “On the time required to recognize properties of graphs: a problem”. In: *SIGACT News* 5.4 (1973), pp. 15–16 (cit. on p. 337).

[Rot82] Alvin E. Roth. “The Economics of Matching: Stability and Incentives”. In: *Math. Oper. Res.* 7.4 (1982), pp. 617–628 (cit. on p. 335).

[RSW22] Mohammad Roghani, Amin Saberi, and David Wajc. “Beating the Folklore Algorithm for Dynamic Matching”. In: *ITCS*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 111:1–111:23 (cit. on p. 335).

[RV75] Ronald L. Rivest and Jean Vuillemin. "A Generalization and Proof of the Aanderaa-Rosenberg Conjecture". In: *STOC*. ACM, 1975, pp. 6–11 (cit. on p. 337).

[RV76] Ronald L. Rivest and Jean Vuillemin. "On Recognizing Graph Properties from Adjacency Matrices". In: *Theor. Comput. Sci.* 3.3 (1976), pp. 371–384 (cit. on p. 337).

[RWZ20] Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. "Vector-Matrix-Vector Queries for Solving Linear Algebra, Statistics, and Graph Problems". In: *APPROX-RANDOM*. Vol. 176. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 26:1–26:20 (cit. on pp. 338, 349).

[Ten02] Moshe Tennenholtz. "Tractable combinatorial auctions and b-matching". In: *Artif. Intell.* 140.1/2 (2002), pp. 231–243 (cit. on p. 335).

[Vai89] Pravin M. Vaidya. "A New Algorithm for Minimizing Convex Functions over Convex Sets (Extended Abstract)". In: *FOCS*. IEEE Computer Society, 1989, pp. 338–343 (cit. on pp. 339, 345, 347).

[VWW20] Santosh S. Vempala, Ruosong Wang, and David P. Woodruff. "The Communication Complexity of Optimization". In: *SODA*. SIAM, 2020, pp. 1733–1752 (cit. on p. 339).

[Yao88] Andrew Chi-Chih Yao. "Monotone Bipartite Graph Properties are Evasive". In: *SIAM J. Comput.* 17.3 (1988), pp. 517–520 (cit. on p. 337).

[Zha04] Shengyu Zhang. "On the Power of Ambainis's Lower Bounds". In: *ICALP*. Vol. 3142. Lecture Notes in Computer Science. Springer, 2004, pp. 1238–1250 (cit. on pp. 335–337).

Paper G

Incremental $(1-\varepsilon)$-Approximate Dynamic Matching in $O(\mathrm{poly}(1/\varepsilon))$ Update Time

Joakim Blikstad, Peter Kiss

Article published in 31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands. [BK23]
Full version at `https://arxiv.org/abs/2302.08432`.

Abstract

In the dynamic approximate maximum bipartite matching problem we are given bipartite graph G undergoing updates and our goal is to maintain a matching of G which is large compared the maximum matching size $\mu(G)$. We define a dynamic matching algorithm to be α (respectively (α, β))-approximate if it maintains matching M such that at all times $|M| \geq \mu(G) \cdot \alpha$ (respectively $|M| \geq \mu(G) \cdot \alpha - \beta$).

We present the first deterministic $(1 - \varepsilon)$-approximate dynamic matching algorithm with $O(\text{poly}(\varepsilon^{-1}))$ amortized update time for graphs undergoing edge insertions. Previous solutions either required super-constant [Gupta FSTTCS'14, Bhattacharya-Kiss-Saranurak SODA'23] or exponential in $1/\varepsilon$ [Grandoni-Leonardi-Sankowski-Schwiegelshohn-Solomon SODA'19] update time. Our implementation is arguably simpler than the mentioned algorithms and its description is self contained. Moreover, we show that if we allow for additive $(1, \varepsilon \cdot n)$-approximation our algorithm seamlessly extends to also handle vertex deletions, on top of edge insertions. This makes our algorithm one of the few small update time algorithms for $(1 - \varepsilon)$-approximate dynamic matching allowing for updates both increasing and decreasing the maximum matching size of G in a fully dynamic manner.

Our algorithm relies on the weighted variant of the celebrated Edge-Degree-Constrained-Subgraph (EDCS) datastructure introduced by [Bernstein-Stein ICALP'15]. As far as we are aware we introduce the first application of the weighted-EDCS for arbitrarily dense graphs. We also present a significantly simplified proof for the approximation ratio of weighed-EDCS as a matching sparsifier compared to [Bernstein-Stein], as well as simple descriptions of a fractional matching and fractional vertex cover defined on top of the EDCS. Considering the wide range of applications EDCS has found in settings such as streaming, sub-linear, stochastic and more we hope our techniques will be of independent research interest outside of the dynamic setting.

G.1 Introduction

Matchings are fundamental objects of combinatorical optimization with a wide range of practical applications. The first polynomial time algorithm for finding a maximum matching was published by Kuhn[1] [Kuh55] in 1955 which ran in $O(n^4)$ time. A long line of papers have focused on improving this polynomial complexity. Notably Edmonds and Karp [EK72] showed the first $O(n^3)$ time algorithm for the problem which was later improved to $O(n^{2.5})$ [HK73]. Mucha and Sankowski [MS04] showed maximum matching can be solved in matrix multiplication time, that is in $O(n^\omega)$ where ω is currently around 2.38. In the very recent breakthrough result of Chen-Kyng-Liu-Peng-ProbstGutenberg-Sachdeva [CKLPGS22], they showed an $O(m^{1+o(1)})$ time algorithm for the maximum flow problem (which generalizes bipartite matching) providing the first near-linear time algorithm, essentially settling the problem in the sequential setting.

Dynamic Model. This paper focuses on the matching problem in the dynamic model where it has received extensive research attention in recent years, see e.g [BK22; Waj20; ACCSW18; BK21; BS16; PS16; AAGPS19; BFH21; CS18; NS16; San07; BHN16] and many more. In this setting our task is to maintain a large matching as the graph undergoes updates. We will refer to updates being fully dynamic if they concern both insertions and deletions and partially dynamic if only one of the two is allowed. The objective is to minimize the time spent maintaining the output after each update. Throughout the paper we will always refer to update time in the amortized sense—averaged over the sequence of updates. In [San07] Sankowski has shown the fist improvement for the fully dynamic maximum matching problem in terms of update time ($O(n^{1.45})$) compared to static recomputation after each update. Unfortunately, based on well accepted hardness conjectures no dynamic algorithm for the maximum matching problem may achieve update time sub-linear in n [HKNS15]. Most works focused on the relaxed approximate version of the problem where the goal is to maintain a large matching in G with respect to the maximum matching size $\mu(G)$. We will refer to matching algorithms as α-approximate (respectively (α, β)-approximate) if it maintains matching M such that at all times $|M| \geq \mu(G) \cdot \alpha$ (respectively $|M| \geq \mu(G) \cdot \alpha - \beta$).

Fully Dynamic Approximate Matching. The holy grail of dynamic algorithm design is to achieve an update time of $O(\text{polylog}(n))$ or ideally even constant. For the fully dynamic approximate matching problem, a long line of research [BGS18; BHN17; BHI18; BK19; BDHSS19; OR10; Sol16; BCH17] has lead to algorithms achieving $\approx \frac{1}{2}$-approximation with $O(\text{polylog}(n))$ and constant update time. No fully dynamic algorithm has been found achieving better than $\frac{1}{2}$-approximation in $O(\text{polylog}(n))$ update time for the problem, and this challenge appears to be one of

[1] However, this result is usually attributed to Kőnig and Egerváry.

the most celebrated problem within the dynamic matching literature. A set of interesting papers focused on $\approx \frac{2}{3}$-approximation in $\tilde{O}(\sqrt{n})$ update time [BS15; BS16; Kis22; GSSU22] and other approximation-ratio to polynomial-update-time trade-offs in the better-than-$\frac{1}{2}$-approximation regime were achieved by [BK22; BLM20; RSW22]. Note that through periodic recomputation of the matching (roughly every $\varepsilon\mu(G)$ updates) we can achieve fully dynamic $(1-\epsilon)$-approximation in $\tilde{O}(n)$ update time [GP13]. Very recently [BRR23] has shown that $(1-\varepsilon)$-approximation is possible in slightly sublinear update time $O(n/\log^*(n)^{O(1)})$ suggesting that there might exist efficient non-trivial algorithms for the problem. Note that very recently [Beh23; BKSW23] have independently shown that if our goal is to maintain the *size* of the maximum matching (and not the edge-set) then sub-$\frac{1}{2}$ approximation is achievable in polylogarithmic update time.

Partially Dynamic Matching Algorithms. For small approximation ratios, achieving polylogarithmic update time for fully dynamic matching seems far out of reach with current techniques, or perhaps even impossible. Hence, a long line of papers have focused on maintaining a $(1-\varepsilon)$-approximate matching in partially dynamic graphs: either incremental (edge insertions) or decremental (edge deletions). The first $O(\text{poly}(\log(n), \varepsilon^{-1}))$ algorithm for maintaining a $(1-\varepsilon)$-approximate matching under edge insertions is due to Gupta [Gup14], with amortized update time $O(\log^2(n)/\varepsilon^4)$. Their algorithm models the bipartite matching problem as a linear program, and uses the celebrated multiplicative-weights-updates framework. Generalizing the result of [Gup14] recently Bhattacharya-Kiss-Saranurak [BKS23a] has shown that an arbitrarily close approximation to the solution of a linear program undergoing updates either relaxing or restricting (but not both) its solution polytope can be maintained in $O(\text{poly}(\log n, \varepsilon^{-1}))$ update time. Hence, the algorithm of [BKS23a] shows how to maintain a $(1-\varepsilon)$-approximate matching under either decremental or incremental updates with a unified approach. As both of these papers rely on static linear program solver sub-routines their update times inherently carry $\log(n)$ factors, and it seems implausible that these techniques can achieve constant update time independent of n.

The decremental algorithms of [BPS20; JJST22] focus on maintaining "evenly spread out" fractional matchings so that they are robust against edge-deletions. These algorithms rely on either maximum-flow computation or convex optimization sub-routines which similarly to LP-solvers carry $\log(n)$-factors into the update time.

The first constant time[2] partially dynamic matching algorithm is due to [GLSSS19] who solve $(1-\varepsilon)$-approximate matching in incremental graphs with an update time of $(1/\varepsilon)^{O(1/\varepsilon)}$. Their solution relies on augmenting path elimination, a technique used commonly for the matching problem in the static setting. However, enumerating augmenting paths seems to inherently carry an exponential dependency on $1/\varepsilon$ due to the number of possible such paths present in the graph.

[2]That is constant time when ε is constant, i.e. the update time should be independent of n.

As far as we are aware partially dynamic $(1-\epsilon)$-approximate matching algorithms with update time independent of n and faster than some exponential in ϵ are all restricted to a relaxed version of the problem where the graph may undergo vertex insertions/deletions. Such an algorithm can simply be obtained through straightforward periodic rebuilds (if we allow for additive $\epsilon \cdot n$ slack) or as shown by Zheng-Henzinger [ZH23][3]. Hence, it we can observe the following apparent gap in the literature of partially dynamic matching algorithms:

Question G.1.1. *Can we maintain a $(1-\varepsilon)$-approximate maximum matching of a partially dynamic bipartite graph in $O(\text{poly}(\varepsilon^{-1}))$ update time?*

Based on the current landscape of the dynamic algorithms literature, achieving $(1-\varepsilon)$-approximation under fully dynamic updates in small update times seems to be far out of reach. Contrary to the extensive research effort, no fully dynamic algorithm with $\text{poly}(\log(n), \varepsilon^{-1})$ has been found for maintaining matchings with better than $\frac{1}{2}$-approximation. This apparent difficulty proposes the research of dynamic models somewhere between fully and partially dynamic updates. The previously mentioned paper by Zheng-Henzinger [ZH23] implements a $(1-\epsilon)$-approximate algorithm with $O(1/\varepsilon)$-update time which can support vertex insertions and deletions on separate sides of the bipartition. The existence of this new result proposes the following natural question:

Question G.1.2. *Under what kind of non-partially dynamic updates can we maintain a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph?*

G.1.1 Our Contribution

In this paper we provide a positive answer to **Question 1** and make progress towards understanding **Question 2**. Our main result is the first $O(\text{poly}(1/\varepsilon))$ update time $(1-\varepsilon)$-approximate dynamic matching algorithm for bipartite graphs undergoing edge insertions:

Theorem G.1.3. *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph undergoing edge insertions in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

Previous algorithms for $(1-\varepsilon)$ approximate dynamic matching under edge updates required update times which were either super-constant [Gup14; BKS23a] or had an exponential dependency on ε^{-1} [GLSSS19]. Furthermore, our algorithm

[3] A very recent paper of Zheng-Henzinger [ZH23] has initially claimed an algorithm which can maintain a $(1-\epsilon)$-approximate matching in $O(1/\varepsilon)$ update time under edge deletions. However, the authors have found a mistake in their paper and claim that their algorithm only works under specific vertex updates

is arguably simpler then previous implementations and it is self contained (except for the static computation of $(1-\varepsilon)$-approximate maximum matchings) where as most dynamic matching algorithms either rely on heavy machinery from previous papers or use black-box tools like multiplicative weight updates or flow-subroutines.

We further show that if we allow for (additive) $(1, \varepsilon \cdot n)$-approximation[4] our algorithm seamlessly extends to a wider range of updates:

Theorem G.1.4. *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1, \epsilon \cdot n)$-approximate maximum matching of a bipartite graph undergoing edge insertions and vertex deletions in total update time of $O(n/\varepsilon^8 + m/\varepsilon^5)$.*

In contrast to the similar update time result of [ZH23] which allows for edge deletions and vertex insertions on one side of the bipartition our algorithm allows from arbitrary vertex deletions. Our algorithm maintains a $(1-\varepsilon)$-approximate maximum matching of the graph throughout updates which can both increase and decrease the maximum matching size of the graph. Hence, we hope our techniques provide useful insight towards fully dynamizing $(1-\varepsilon)$-approximate algorithms for the matching problem.

Our algorithm relies on the weighted variant of the celebrated Edge-Degree-Constrained-Subgraph (EDCS) matching sparsifier. The unweighted EDCS (first introduced by Bernstein-Stein [BS15]) has found applications in a number of different computational settings: streaming [Ber20; ABBMS19; AB21], stochastic, one way communication, fault tolerant [AB19], sub-linear [Beh23; BKSW23; BRR23; BKS23b] and dynamic [BS15; BS16; Kis22; GSSU22; Beh23]. On the other hand the *weighted* EDCS variant which provides a tighter approximation has only found applications in small arboricity graphs [BS15]. Hence, we initiate the study of the weighted EDCS in dense graphs.

Furthermore, we show a significantly simplified proof for the approximation ratio of the weighted EDCS datastructure with respect to the maximum matching size. In our proof, we identify simple and explicit descriptions of a fractional matching and fractional vertex covers defined on top of the weighted EDCS, which might be of independent interest. Moreover, we show that the dependence on the slack parameter on the maximum degree of the weighted EDCS is exactly quadratic. This in sharp contrast to the unweighted EDCS where the same relationship have been proven to be linear [Beh21]. While within the dynamic matching algorithm literature papers don't tend to focus on exact ϵ complexities but rather n dependence, in models such as semi-streaming and distributed the ϵ dependency usually gains more focus. Our hard example (most likely) rules out applications of weighted EDCS in these models for obtaining sub ϵ^{-2} round/pass complexity algorithms. We hope that these observations will be of independent research interest due to the wide-spread popularity of the EDCS datastructure for solving matching problems.

[4] Recall that this means that we maintain a matching of size at least $\mu - \varepsilon n$, as opposed to $\mu - \varepsilon\mu$, where μ denotes the size of the maximum matching.

G.1.2 Our Techniques

Assume H is a multi-graph defined on the vertex set of G and let $\deg_H(v)$ stand for the degree of vertex v in H. We define the degree of an edge to be the sum of the degrees of its endpoints.[5]

Definition G.1.5 (Weighted EDCS [BS15])**.** Given a graph $G = (V, E)$, a multiset H is called a weighted EDCS with parameter β if[a]:

(i) $\deg_H(u) + \deg_H(v) \leq \beta$ for all edges $(u, v) \in H$.

(ii) $\deg_H(u) + \deg_H(v) \geq \beta - 1$ for all edges $(u, v) \in E$.

If H is not a weighted EDCS, we call an edge $e \in H$ *overfull* if it violates (i), and an edge $e \in E$ *underfull* if it violates (ii).

[a]Some authors use an additional parameter $\beta^- < \beta$ which replaces the "$\beta - 1$" in the degree-constraint. For our purposes, we will always have $\beta^- = \beta - 1$.

If $\beta = \Omega(\varepsilon^{-2})$ and H is a β-WEDCS of G then $\mu(H) \geq \mu(G) \cdot (1 - \varepsilon)$ ([BS15], Theorem G.5.1). In order to derive our incremental result we show that a β-WEDCS can be efficiently maintained greedily under edge insertions. In turn we can efficiently maintain a $(1 - \epsilon)$-approximate matching within the support of H through periodic recomputation.

Define a valid update of H to be one of the following: (i) and edge $e \in H$ which is overfull with respect to H gets deleted from H; and (ii) a copy of an edge $e \in E$ which is underfull with respect to H is added to H. In Lemma G.3.3 (slightly improving on the similar lemma's of [AB19; Ber20; BS15]) we show that if H is initialized as the empty graph and only undergoes valid updates, then it there are at most $O(\mu(G) \cdot \beta^2)$ many updates.

Fix some $\beta = \Theta(\varepsilon^{-2})$. Assume G is initially empty and initialize H to be an empty edge set (note that by definition initially H is a β-WEDCS of G). Assume edge e is inserted into G. If at this point e is not underfull with respect to H there is nothing to be done as H remained a valid WEDCS of G. If e is underfull with respect to H we add copies of it to H until it is not. This process of adding e to H has increased the edge degree of edges neighbouring e in H and some of them might have became overfull. To counteract this we iterate through the neighbours of e in an arbitrary order and if we find an overfull edge e' we remove it from H. This edge removal decreases the edge degrees in the neighbourhood of e'. To counteract this we recurse and look for underfull edges in the neighbourhood of e'. If such an edge e'' is found we add copies of it to H until e'' is not underfull and repeat the same

[5]Note that we are sticking to the notation weighted-EDCS instead of multi-EDCS to be in line with the naming convention of [BS15] which defined H to be a weighted graph with integer edge weights.

steps as if e'' was just inserted into G. This defines a natural recursive process for restoring the WEDCS properties after each edge insertion in a local and greedy way.

Whenever we have to explore the neighbourhood of an edge in $O(\Delta)$ time (where Δ is the max-degree) to either check for underfull or overfull edges we do so because H underwent a valid update. By Lemma G.3.3 this may only happen at most $O(\mu(G) \cdot \beta^2)$ times. Hence, naively the total work spent greedily fixing the WEDCS properties is $O(\mu(G) \cdot \Delta \cdot \beta^2)$. For some graphs this value might be significantly larger then m. In order to improve the update time to $O_\beta(m)$ we assign a counter c_v to each vertex v measuring the number of valid updates of H the neighbourhood of v has underwent. Once c_v grows to $\Omega(\beta^2 \cdot \varepsilon^{-1})$ we mark v dirty and ignore further edges inserted in the neighbourhood of v. By marking a single vertex dirty and ignoring some edges incident on it we may loose out only on a single edge of any maximum matching. However, whenever we mark a vertex dirty we can charge $\Omega(\beta^2 \cdot \varepsilon^{-1})$ valid updates of H to that vertex. As there may be at most $O(\mu(G) \cdot \beta^2)$ valid updates of H in total we may only mark $O(\mu(G) \cdot \varepsilon)$ vertices dirty hence we will only ignore an $O(\epsilon)$-fraction of any maximum matching within the graph through ignoring edges incident on dirty vertices. As we may scan the neighbourhood of vertex v at most $O(\beta^2 \cdot \varepsilon^{-1}) = \text{poly}(\varepsilon^{-1})$ times until v is marked dirty we ensure that each edge is explored $\text{poly}(\varepsilon^{-1})$ times guaranteeing a total running time of $O(m \cdot \text{poly}(\varepsilon^{-1}))$. Full details can be found in Section G.3.

Towards Full Dynamization. The algorithm almost seamlessly adopts to vertex deletions if we allow for $(1, \varepsilon \cdot n)$-approximation[6]. Whenever a vertex gets deleted from the graph our WEDCS H might be locally affected. This means that over the full run of the algorithm, H may undergo further valid updates then the $O(\mu(G) \cdot \beta^2)$ bound provided by Lemma G.3.3. A potential function based argument allows us to claim that each vertex deletion may increase the total number valid updates H may undergo by $O(\beta^2)$. As each vertex may be deleted at most once this means that the total number of valid updates we might make to restore H is $O(n \cdot \beta^2)$, each update requiring $O(\Delta)$ time if naively implemented. By marking vertices as dirty as before we can guarantee amortized $O(\text{poly}(1/\varepsilon))$ update time. However, now we must mark up to $\approx \varepsilon \cdot n$ vertices as dirty (as opposed to $\approx \varepsilon \cdot \mu(G)$ like before), which means we may miss out on $\varepsilon \cdot n$ edges of the maximum matching.

G.2 Preliminaries

Matching Notation. Let $N_E(v)$ stand for the edges neighbouring vertex v in E. A fractional matching f of a graph G is an assignment of the edges of G to values in the range $[0, 1]$ such that for all vertices $v \in V$ it holds that $\sum_{e \in N_E(v)} f_e \leq 1$.

[6] Readers may reasonable argue that the additive slack is not necessary as a number of vertex-sparsification techniques exist in literature allowing us to improve the approximation to purely multiplicative slack in the dynamic setting. Unfortunately, these techniques don't appear to be robust against vertex-wise updates.

The *size* of a fractional matching is simply the sum of the fractional values over its edges. That is a maximum fractional matching is the solution to the linear program $\max\{\sum_{e\in E} f_e : \sum_{e\in N_E(v)} f_e \leq 1 \text{ for all } v \in V, f \geq 0\}$. A solution x to the dual of this program $\min\{\sum_{v\in V} x_v : x_u + x_v \geq 1 \text{ for all } (u,v) \in E, x \geq 0\}$ is a fractional vertex cover.

Approximation with respect to a fractional matching is defined simmilarly as with respect to integral matchings. For a graph $G = (V, E)$ we use $\mu(G)$ to denote the size of the maximum matching in G. Likewise, we use $\mu^*(G)$ to denote the size of the maximum *fractional* matching. It is well-known that $\mu(G) \leq \mu^*(G) \leq \frac{3}{2}\mu(G)$ for any graph, and that $\mu^*(G) = \mu(G)$ in bipartite graphs.

Theorem G.2.1 (Hopcroft-Karp [HK73])**.** *There exists a deterministic static algorithm which finds a $(1-\varepsilon)$-approximate maximum matching of a graph G on m edges in $O(m/\varepsilon)$ time.*

G.3 Incremental Approximate Matching

We start by showing our incremental fractional matching algorithm, and then show how to extend it (for bipartite graphs) to also maintain an integral matching.

G.3.1 Weighted EDCS & Fractional Matchings

In this section we show our algorithm to (almost[7]) maintain a weighted EDCS H in an incremental graph. It is well-known that such an H will be a $(1-\varepsilon)$-matching sparsifier on *bipartite* graphs, that is a "sparse" subgraph with $\mu(H) \geq (1-\varepsilon)\mu(G)$ [BS15].

As we show later in Section G.5.1 (Theorem G.5.1), we even known something stronger: there is an explicit fractional matching in H of size at least $(1-\varepsilon)\mu^*(G)$, defined as

$$f_{(u,v)} = \min\left(\frac{1}{\deg_H(v)}, \frac{1}{\deg_H(u)}\right) \text{ on each } (u,v) \in H. \tag{G.1}$$

Note that [BS16] similarly (implicitly) defines a large fractional matching on the support of a weighted EDCS, however our construction and analysis are arguably simpler. This fractional matching is also valid for general (non-bipartite) graphs. Hence our incremental algorithm will also maintain this explicit $(1-\varepsilon)$-approximate fractional matching (even in non-bipartite graphs). Formally we prove the following theorem.

Theorem G.3.1. *For any $\varepsilon \in (0,1)$, there is an algorithm (Algorithm G.1) that maintains a $(1-\varepsilon)$-approximate maximum* fractional *matching in an incremental*

[7] As we will see later in this section, our sparsifier H will be a weighted EDCS for $G \setminus R$, where R is a subgraph of G with very small maximum matching size $\mu(R) = O(\varepsilon\mu(G))$.

graph in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$. Additionally, this fractional matching is always supported on a set of edges H of size $|H| \leq \Theta(\mu(G)/\varepsilon^2)$ and maximum degree $O(1/\varepsilon^2)$.

First we need two standard facts about weighted EDCS. For completeness, we prove these in Section G.7. Lemma G.3.3 has only been shown before for unweighted EDCS [AB19; Ber20; BS15] and not weighted (but the arguments are very similar).

Lemma G.3.2. *In a β-WEDCS H, the maximum degree is at most β and $|H| \leq \beta\mu^*(G)$.*

Lemma G.3.3. *If a multiset of edges H is only ever changed by removing overfull edges and adding underfull edges, then there are at most $\beta^2\mu^*(G)$ such insertions/deletions to H.*

Overview. Our algorithm (see Algorithm G.1) will maintain a weighted EDCS H with $\beta = \Theta(1/\varepsilon^2)$. We also maintain the $(1-\varepsilon)$-approximate fractional matching f as in Equation (G.1) and Theorem G.5.1.

When we get an edge-insertions (u, v), we need to reestablish the property that H is an EDCS. If (u, v) is underfull ($\deg_H(u) + \deg_H(v) < \beta - 1$), we add it (maybe multiple times) to H. This means that $\deg_H(u)$ (similarly $\deg_H(v)$) increases, which can potentially make some incident edge $(u, w) \in H$ overfull ($\deg_H(u) + \deg_H(w) > \beta$), so we must remove one such edge. This might in turn lead to some edge $(w, z) \in E$ being underfull (as now $\deg_H(w)$ decreased), so we add this edge to H. This process continues, so both from u and v we need to search for alternating paths of underfull and overfull edges (as is standard in EDCS-based algorithms). In total, Lemma G.3.3 says there are $O(n/\varepsilon^4)$ updates to H over the full run of the algorithm.

We note that searching for an *overfull* edge is cheap: the maximum degree in H is just $O(\beta)$ (Lemma G.3.2), so we can afford to, in $\Theta(1/\varepsilon^2)$ time, check all incident edges. However, searching for *underfull* edges is more expensive: this time we cannot afford to just go through all neighboring edges in E, as we no longer have a bound on the maximum degree.

To overcome this we use an amortization trick which allows us to ignore a vertex if we touched it too many times. There are only $\beta^2\mu^*(G)$ updates to H in total (Lemma G.3.3), so there will only be $\varepsilon\mu^*(G)$ many vertices incident to more than $2\beta^2/\varepsilon$ of these updates. Any edges incident to these "update-heavy" vertices we may ignore, as this may only decrease the maximum matching size by an ε fraction. We thus only need to check each edge a total of $O(1/\varepsilon^5)$ times over the run of the algorithm, except when it is in H already. Note that H is no longer a weighted EDCS of $G = (V, E)$, but rather of $G' = (V, (E \setminus R) \cup H)$ where R is this set of edges we ignored (with $\mu^*(R) \leq \varepsilon\mu^*(G)$).

Algorithm G.1: Incremental Weighted EDCS & Frasctional Matching

```
// Initially E = H = ∅ and deg_H(v) = visits[v] = 0 for all v ∈ V.
// When an edge insertion e appears, add it to E and call FixEdge(e).
1  function FixEdge(e = (u, v))
2      if deg_H(u) + deg_H(v) > β and (u, v) ∈ H then          // overfull
3          Remove (one copy of) the edge (u, v) from H
4      if deg_H(u) + deg_H(v) < β − 1 then                     // underfull
5          Add (one copy of) the edge (u, v) to H
6      if the edge was added or removed then
7          Update deg_H(u), deg_H(v), and the fractional matching accordingly
8          FixVertex(u), FixVertex(v)
9  function FixVertex(v)
10     visits[v] ← visits[v] + 1
11     if visits[v] < 2β²/ε then
12         for edge e ∈ E incident to v do
13             FixEdge(e)
14     else
15         for edge e ∈ H incident to v do
16             FixEdge(e)
```

Running Time. We analyse the total update time spent in different parts of our algorithm.

- We first note that FixEdge runs in constant time whenever it does not update H. It is called once for each edge-insertion (total $O(m)$ times), and also some number of times from FixVertex.

- Now consider the case when FixEdge does update H (which happens at most $\beta^2\mu^*(G)$ times per Lemma G.3.3). Now the algorithm uses $O(\beta)$ time for the update of the fractional matching and insertion/removal in H, in addition to exactly two calls to FixVertex. Except for these calls to FixVertex, over the run of the algorithm we hence spend a total of $O(\mu(G)\beta^3) = O(n/\varepsilon^6)$ time.

- By the previous point, we will call FixVertex at most $2\beta^2\mu^*(G)$ times. In each call, we either loop through all incident edges in H or E. If we loop through H, we visit β many edges (by Lemma G.3.2). Either these FixEdge calls take constant time, or they are already accounted for in the previous

point. In total, this thus accounts for another $O(\mu(G)\beta^3) = O(n/\varepsilon^6)$ running time.

- We account for the case when FixVertex loops through all incident edges in E differently. Consider how often a specific edge e appears in the for-loop at line 13. Each endpoint vertex of e will reach this line at most $O(\beta^2/\varepsilon)$ times. Hence, in total for all edges, line 13 is run at most $O(m\beta^2/\varepsilon) = O(m/\varepsilon^5)$ times.

Approximation Guarantee. We now argue the approximation ratio. We will show that the fractional matching supported on H is a $(1-2\varepsilon)$-approximation of maximum fractional matching in G. If one want a $(1-\varepsilon')$-approximation, then one can run the algorithm in the same asymptotic update time setting $\varepsilon = \varepsilon'/2$, and changing β accordingly.

Define R_V to be the set of "dirty"/"update-heavy" vertices: that is vertices v for which FixVertex(v) has been called at least $2\beta^2/\varepsilon$ many times (i.e. $\mathsf{visits}[v] \geq 2\beta^2/\varepsilon$). By a counting argument $|R_V| \leq 2\beta^2\mu^*(G)/(2\beta^2/\varepsilon) = \varepsilon\mu^*(G)$, since by Lemma G.3.3 in total there are only $\beta^2\mu^*(G)$ many updates to H, each issuing exactly two calls to FixVertex. If R_E is the set of edges incident to R_V, then $\mu^*(R_E) \leq |R_V| \leq \varepsilon\mu^*(G)$ since R_V is a vertex cover of R_E.

Define $G' = (V, E \setminus (R_E \setminus H))$. By the above, $\mu^*(G') \geq (1-\varepsilon)\mu^*(G)$. We will finish the proof by arguing that H is a weighted EDCS of G', and thus that our fractional matching is of value at least $(1-\varepsilon)\mu^*(G') \geq (1-\varepsilon)^2\mu^*(G) \geq (1-2\varepsilon)\mu^*(G)$. Whenever an edge is added or removed to H, we call FixVertex on its endpoints, and no other degrees $\deg_H$ have changes. Every time FixVertex(v) is called for $v \notin R_V$, we make sure that all edges $e \in E$ incident to it satisfy the definition of an EDCS, and when FixVertex(v) is called for some $v \in R_v$, we check the edges incident to H. We note that when a vertex becomes "update-heavy" (added to R_v), then we do **not** immediately remove all incident edges from H (as then we no longer have the same bound on the number of updates to H since Lemma G.3.3 no longer applies).

Remark G.3.4. We note that our algorithm runs in time $O(n\beta^3 + m\beta^2/\varepsilon)$, and a valid question is whether setting $\beta = \Theta(1/\varepsilon^2)$ is actually necessary? Recently it was shown that for unweighted EDCS $\beta = \Theta(1/\varepsilon)$ is enough [Beh21]. However, for weighted EDCS the ε^2 dependency is indeed necessary, as we show by an example where this is asymptotically tight in Section G.5.2 (Theorem G.5.4).

G.3.2 Integral Matchings in Bipartite Graphs

In this section we argue how to extend our fractional matching algorithm (Theorem G.3.1) to maintain an integral matching instead (for bipartite graphs), in the same asymptotic update time. We cannot use known dynamic rounding techniques [BKS23a; Waj20], since all these incur polylog(n) factors or require randomization, and we are aiming for update time independent of n. In fact, our technique is simple

and combinatorial; and only relies on the standard Hopcroft-Karp algorithm for finding a $(1-\varepsilon)$-approximate matching in the static setting [HK73].

Theorem G.1.3. *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph undergoing edge insertions in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

Remark G.3.5. Before proving the above theorem, we briefly explain how to achieve a slightly less efficient version (amortized $O(1/\varepsilon^8)$ update time) by using the fully dynamic $(1-\varepsilon)$-approximate matching algorithm of Gupta-Peng [GP13] as a black box. The idea is to run the fully dynamic algorithm on our sparsifier—the weighted EDCS H. This way we maintain a matching M of size $|M| \geq (1-\varepsilon)\mu(H) \geq (1-\varepsilon)^2\mu(G) \geq (1-2\varepsilon)\mu(G)$.

Gupta-Peng [GP13] state that their algorithm runs in $O(\sqrt{m}/\varepsilon^2)$ time per update. However, as previously pointed out by e.g. [BS15, Lemma 1], it is in fact more efficient when the max-degree Δ is low, in which case the update time is only $O(\Delta/\varepsilon^2)$. Since H always has max-degree $\beta = \Theta(1/\varepsilon^2)$, we can maintain the integral matching M in $O(1/\varepsilon^4)$ time *per update to* H. Over the run of the algorithm, we only perform $O(\mu(G)/\varepsilon^4)$ updates to H (see Lemma G.3.3), hence the total additional update time spent maintaining the integral matching will be $O(n/\varepsilon^8)$.

Proof of Theorem G.1.3. To prove Theorem G.1.3, we need a slightly more refined analysis than the one above. We still run our incremental algorithm (Algorithm G.1 and Theorem G.3.1) to maintain a weighted EDCS H together with a fractional matching supported on H. Similarly to above, we additionally maintain an $(1-\varepsilon)$-approximate (integral) matching M of H.

The main idea of the fully dynamic algorithm of Gupta-Peng [GP13] is to lazily recompute (in $O(|H|/\varepsilon)$ time via Hoproft-Karp Theorem G.2.1) M every $\approx \varepsilon\mu$ updates to H (indeed, during this few updates, the matching size cannot change its value by more than $\varepsilon\mu$). There are two observations which helps us to do better:

(i) The graph G (but not the sparsifier H) is incremental, so $\mu(G)$ can only grow.

(ii) We know a good estimate of $\mu(G)$, namely the size of our fractional matching. Denote by $\tilde{\mu}$ the value of the maintained fractional matching, so that $(1-\varepsilon)\mu(G) \leq \tilde{\mu} \leq \mu(G)$.

The above two observations mean that we only need to recompute the matching M whenever $\mu(G)$ actually have increased significantly (namely by a $(1+\Theta(\varepsilon))$-factor), and not just every $\varepsilon\mu$ updates.

Formally, whenever $|M| \geq (1-\varepsilon)^2\tilde{\mu}$ we know that M is still a $(1-3\varepsilon)$-approximation since then $|M| \geq (1-\varepsilon)^2\tilde{\mu} \geq (1-\varepsilon)^3\mu(G) \geq (1-3\varepsilon)\mu(G)$. Conversely, whenever $|M| < (1-\varepsilon)^2\tilde{\mu}$, we recompute M in time $O(|H|/\varepsilon)$ (Theorem G.2.1) so that it is a $(1-\varepsilon)$-approximation of the maximum matching in H (and thus also a $(1-2\varepsilon)$-approximation of the maximum matching in G).

Let us now bound the total time spent recomputing M. Let $M_1, M_2, \ldots, M_t$ be the different approximate matchings we compute during the run of the algorithm. We first note that at the time when we compute M_{i+1}:

$$|M_i| \leq (1-\varepsilon)^2\tilde{\mu} \leq (1-\varepsilon)\left((1-\varepsilon)\mu(H)\right) \leq (1-\varepsilon)|M_{i+1}| \tag{G.2}$$

This in turn means that $|M_i| \leq (1-\varepsilon)^{t-i}n$ (since $|M_t| \leq n$), and hence that $\sum_{i=1}^{t}|M_i| \leq n\sum_{i=0}^{\infty}(1-\varepsilon)^i \leq n/\varepsilon$, by a geometric sum.

Finally we note that we spend $O(|M_i|/\varepsilon^3)$ time in order to compute M_i. Indeed, when we compute M_i, we did so in $O(|H|/\varepsilon)$ time, and $|H| = O(\mu(G)/\varepsilon^2)$ by Lemma G.3.2. This means that in total, over the run of the algorithm, we spend $O(\sum|M_i|/\varepsilon^3) = O(n/\varepsilon^4)$ time maintaining the integral matchings M_i. This is in addition to the time spent maintaining the weighted EDCS H and the fractional matching (see Theorem G.3.1). This concludes the proof of Theorem G.1.3. □

G.4 Vertex Deletions

In this section we will observe that our algorithm can also handle vertex deletions (simultaneously to handling edge insertions) in similar total update time. However, this comes with one caveat: we instead get additive approximation error proportional to εn (that is we maintain a matching of size $\mu^*(G) - \varepsilon n$, instead of $\mu^*(G) - \varepsilon\mu^*(G)$ as before).

Theorem G.4.1. *For any $\varepsilon \in (0,1)$, there is an algorithm that maintains a* fractional *matching of size at least $\mu^*(G) - \varepsilon n$ in an graph G undergoing edge insertions and vertex deletions. The total update time is $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

Proof. The algorithm (Algorithm G.1) remains the same as in Section G.3.1. When a vertex is deleted, we simply remove all it's incident edges from E and H, and call FixVertex on all neighboring vertices (in H) who now changed their degree. The only thing which changes in the analysis is the $\beta^2\mu^*(G)$-bound on the number of updates to H (Lemma G.3.3), which no longer applies. However, we can still get a weaker version of Lemma G.3.3 with a $\beta^2 n$ total update bound instead:

Lemma G.4.2. *If a multiset of edges H is only ever changed by (i) removing overfull edges, (ii) adding underfull edges, and (iii)* removing all edges incident to a vertex when no edges are underfull or overfull*, then there are at most $3\beta^2 n$ insertions/deletions to H.*

Given Lemma G.4.2 (which we prove in Section G.7), we see that the running time analysis of Algorithm G.1 can remain exactly the same! In the approximation guarantee analysis, we now have more "update-heavy" vertices $|R_V| \leq 6\beta^2 n/(2\beta^2/\varepsilon) = 3\varepsilon n$, which is why we now can lose up to $O(n\varepsilon)$ edges from the matching. Otherwise, the approximation guarantee analysis remains the same, and so does the rest of the analysis of the algorithm. □

Rounding in Bipartite Graphs. Similar as in Section G.3.2, we can round the fractional matching to an integral one in bipartite graphs, also while supporting edge-insertions and vertex-deletions simultaneously.

Theorem G.1.4. *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1, \epsilon \cdot n)$-approximate maximum matching of a bipartite graph undergoing edge insertions and vertex deletions in total update time of $O(n/\varepsilon^8 + m/\varepsilon^5)$.*

Proof. Unlike in Section G.3.2, we cannot argue that $\mu(G)$ is increasing when we have vertex deletions. So instead we resort to the Gupta-Peng [GP13] framework discussed in Remark G.3.5 (together with Lemma G.4.2), which has the additional cost of $O(n/\varepsilon^8)$ total update time to maintain an approximate integral matching on our sparsifier H. □

Remark G.4.3. We note that normal vertex-sparsification techniques (such as the one shown in [Kis22] against oblivious adversaries) do not apply here in order to assume $n = \tilde{\Theta}(\mu(G))$ so that the additive error becomes multiplicative again. This is because vertex deletions in the original graph might become edge deletions in the vertex-sparsified graph. We also note that one can achieve similar guarantees of supporting vertex deletions with additive εn slack using *any* edge-incremental algorithm (also for non-bipartite graphs) as a black-box: see Section G.6 for a discussion on how this can be done. The general approach in Section G.6 will give worse dependency on $1/\varepsilon$ (for dense graphs), compared to Theorems G.1.4 and G.4.1 above.

G.5 Tight Bounds of the of Approximation Ratio of a Weighted EDCS

G.5.1 Explicit Fractional Matching

In this section, we give explicit formulas only based on the degrees in H, for a $(1-\varepsilon)$-approximate fractional matching. We prove this by also providing an explicit approximate fractional vertex cover, and showing that these satisfy approximate complimentary slackness. This also significantly simplifies the previous proof [BS15] that H is $(1-\varepsilon)$-matching sparsifier in bipartite graphs.

Theorem G.5.1. *Suppose H is a weighted EDCS of a graph G, with parameter $\beta \geq 25/\varepsilon^2$. Let $f_{(u,v)} = \min\{\frac{1}{\deg_H(v)}, \frac{1}{\deg_H(u)}\}$ for each[8] $(u,v) \in H$. Then f is a $(1-\epsilon)$-approximate fractional matching of G.*

[8]We note that edges e appearing multiple time in H all contribute towards f_e: if e appears ϕ_e times in H, the value of f_e is naturally scaled by ϕ_e.

Define $r_v := \deg_H(v) - \frac{\beta-1}{2}$ for a vertex $v \in H$. We define the fractional matching f as in the statement of the theorem, together with the fractional vertex cover x:

$$f_{(u,v)} = \min\left(\frac{1}{\deg_H(u)}, \frac{1}{\deg_H(v)}\right) \quad \text{for edge } (u,v) \in H \tag{G.3}$$

$$x_v = \begin{cases} \min(1, \frac{1}{2} + \frac{r_v^2}{\beta}) & \text{if } r_v \geq 0 \\ \max(0, \frac{1}{2} - \frac{r_v^2}{\beta}) & \text{if } r_v < 0 \end{cases} \tag{G.4}$$

It is now relatively straightforward (albeit a bit calculation-heavy) to argue that f and x are indeed feasible solutions and that they satisfy approximate complimentary slackness.

Claim G.5.2. *Our f is a fractional matching and our x is a fractional vertex cover of G.*

Proof. Our f is feasible since no vertex v is overloaded by the matching: at most $\deg_H(v)$ many incident edges to v contribute at most $1/\deg_H(v)$ each.

To argue that x is feasible, consider some edge $(u,v) \in E$. Without loss of generality we may assume that $\deg_H(v) \geq \deg_H(u)$ and $\deg_H(v) \geq \frac{\beta-1}{2}$, i.e. $r_v \geq r_u$ and $r_v \geq 0$ (since $\deg_H(u) + \deg_H(v) \geq \beta - 1$ as H is a weighted EDCS). If $r_v^2 \geq \beta/2$, $x_v = 1$ so (u,v) is covered. In the case $r_v^2 < \beta/2$, we instead have $x_v = \frac{1}{2} + \frac{r_v^2}{\beta}$. It is always the case that $x_u \geq \frac{1}{2} - \frac{r_u^2}{\beta}$. Additionally we note that $r_u + r_v \geq 0$ (so $r_u^2 \leq r_v^2$) since $\deg_H(u) + \deg_H(v) \geq \beta - 1$, so we conclude that $x_u + x_v \geq 1$, and hence that (u,v) is covered. □

Claim G.5.3. *The fractional matching f and fractional vertex cover x satisfy $(1 - \frac{3}{\sqrt{\beta}}, 1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta})$-approximate complementary slackness[9]; in particular:*

(i) Whenever $f_{(u,v)} > 0$, then $x_u + x_v \leq 1 + \frac{2}{\sqrt{\beta}} + \frac{1}{\beta}$.

(ii) Whenever $x_v > 0$, then $\sum_{u:(u,v)\in H} f_{(u,v)} \geq 1 - \frac{4}{\sqrt{\beta}}$.

Proof. We verify (i) and (ii).

(i) Suppose $f_{(u,v)} > 0$, then $(u,v) \in H$, so $\deg_H(u) + \deg_H(v) \in \{\beta - 1, \beta\}$. This means that $0 \leq r_v + r_u \leq 1$. If both r_v and r_u are non-negative, we have that $x_u + x_v \leq 2(\frac{1}{2} + \frac{1^2}{\beta}) \leq 1 + \frac{2}{\beta}$. Now, without loss of generality $r_u < 0 \leq r_v$. In case $r_u^2 \geq \beta/2$, we know $x_u = 0$ so $x_u + x_v \leq 1$. In the case when $r_u^2 < \beta/2$,

[9] For completeness, we define the approximate complimentary slackness conditions in Section G.7 and prove them in Lemma G.7.1.

we know $x_u = \frac{1}{2} + \frac{r_u^2}{\beta}$ and $x_v \leq \frac{1}{2} + \frac{r_v^2}{\beta}$. Since $r_u + r_v \leq 1$, we known that $r_v \leq |r_u| + 1$. Concluding:

$$x_v + x_u \leq 1 + \frac{(|r_u|+1)^2 - r_u^2}{\beta} = 1 + \frac{2|r_u|+1}{\beta} < 1 + \frac{2\sqrt{\beta}+1}{\beta} = 1 + \frac{2}{\sqrt{\beta}} + \frac{1}{\beta}.$$

(ii) Suppose $x_v > 0$. Hence $r_v > -\sqrt{\beta/2}$ (else $x_v = 0$), that is $\deg_H(v) > \frac{\beta-1}{2} - \sqrt{\beta/2}$. For any incident edge $(u,v) \in H$, we must have $\deg_H(v) + \deg_H(u) \leq \beta$, so $\deg_H(u) \leq \frac{\beta-1}{2} + (1 + \sqrt{\beta/2})$. Now, we see that we assign weight at least $1/(\frac{\beta-1}{2} + (1 + \sqrt{\beta/2}))$ to the edge (u,v) in f. Since this holds for all the $\deg_H(v) > \frac{\beta-1}{2} - \sqrt{\beta/2}$ incident edges we know that v will in total receive, from the fractional matching f, at least:

$$\sum_{u:(u,v)\in H} f_{(u,v)} \geq \frac{\frac{\beta-1}{2} - \sqrt{\beta/2}}{\frac{\beta-1}{2} + 1 + \sqrt{\beta/2}} = 1 - \frac{\sqrt{8\beta}+2}{\beta + \sqrt{2\beta}+1} \geq 1 - \frac{3}{\sqrt{\beta}}. \qquad \square$$

Proof of Theorem G.5.1. By the above claims and approximate complimentary slackness (see Lemma G.7.1 in Section G.7) we know that $(1 - \frac{3}{\sqrt{\beta}})|x| \leq (1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta})|f|$. Since $(1 - \frac{3}{\sqrt{\beta}})/(1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta}) > 1 - \frac{5}{\sqrt{\beta}}$, we get that $|f| \geq (1 - \frac{5}{\sqrt{\beta}})|x| \geq (1-\varepsilon)|x| \geq (1-\varepsilon)\mu^*(G)$ whenever $\beta \geq 25/\varepsilon^2$. $\square$

G.5.2 Lower Bound

In this section we show that Theorem G.5.1 is tight up to a constant, i.e. that one must set $\beta = \Theta(1/\varepsilon^2)$ in order to guarantee that a weighted EDCS preserves a $(1-\varepsilon)$-fraction of the matching. This might be a bit surprising, considering that for the *unweighted* EDCS, it is known that $\beta = \Theta(1/\varepsilon)$ suffices (to preserve a $(\frac{2}{3} - \varepsilon)$-approximation to the matching [Beh21]).

Theorem G.5.4. *For any $\beta \geq 2$, there exists a (bipartite) graph G together with a weighted EDCS H for which $\mu(H) = (1 - \Theta(1/\sqrt{\beta}))\mu(G)$.*

We show our construction in Figure G.1, and also describe it here formally in words. For simplicity, we will assume that $\beta = 2\gamma^2$ for some integer γ (but it is not difficult to adapt the proof for when β is not twice a square). In our construction, each edge appears at most once in H, and all edges $e \in H$ have $\deg_H(e) = \beta$; all edges $e \in E \setminus H$ have $\deg_H(e) = \beta - 1$.

Define the gadget $G_i = (S_i, L_i, E_i)$ to be a complete bipartite graph in which $|S_i| = \gamma^2 + i$ and $|L_i| = \gamma^2 - i$ (S is for vertices with *small* degree, and L for vertices with *large* degree). The subgraph H will consist of many of these gadgets G_i, so we start by noting a few properties about them. Firstly, vertices in L_i have degree $\frac{\beta}{2} + i$ while those in S_i have degree $\frac{\beta}{2} - i$. This means that any edge in $(u,v) \in E_i$ has degree exactly $\deg_{G_i}(u) + \deg_{G_i}(v) = \beta$. We also note that the maximum matching inside G_i is of size $|L_i| = \gamma^2 - i$.

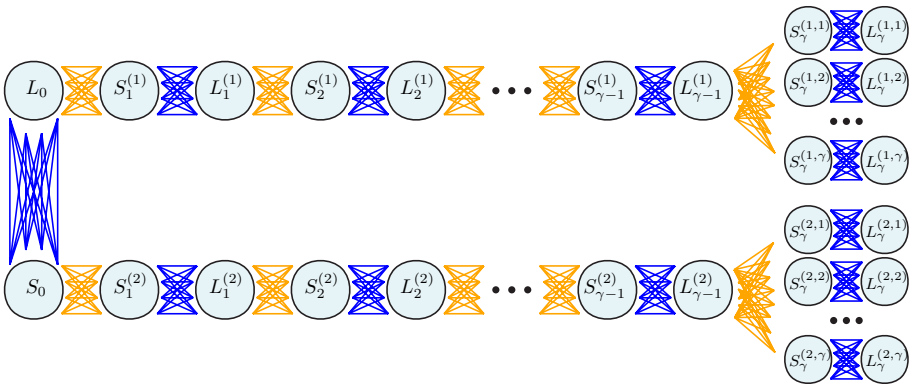


Figure G.1: The lower bound construction. The blue edges are part of H, while the yellow are not. We have $\gamma = \sqrt{\beta/2}$, and each set S_i, L_i indicates an independent set of vertices of size $|S_i| = \gamma^2 + i$ and $|L_i| = \gamma^2 - i$ (so in H they have degrees $\frac{\beta}{2} - i$ and $\frac{\beta}{2} + i$ respectively). The maximum matching in H matches all vertices in L_i to the corresponding S_i. The maximum matching in G however, matches L_i to S_{i+1} (and S_0 to $S_1^{(2)}$), in addition to L_γ which can also be matched to S_γ.

Subgraph H. We begin by describing how the weighted EDCS H looks like, and later we will define what additional edges are also in the full graph G. The subgraph H will exactly consist of:

- One copy of G_0.
- Two copies each of $G_1, G_2, \ldots, G_{\gamma-1}$. Call the copies $G_i^{(1)}$ and $G_i^{(2)}$.
- 2γ copies of G_γ. Call the copies $G_\gamma^{(1,j)}$ and $G_\gamma^{(2,j)}$ for $j = 1, 2, \ldots, \gamma$.

Claim G.5.5. $\mu(H) = 4\gamma^3 - 4\gamma^2 + \gamma$.

Proof. Since the maximum matching size in G_i is $|L_i| = \gamma^2 - i$ we get:

$$\begin{aligned}\mu(H) &= |L_0| + 2\gamma|L_\gamma| + 2(|L_1| + |L_2| + \cdots + |L_{\gamma-1}|) \\ &= \gamma^2 + 2\gamma(\gamma^2 - \gamma) + 2\sum_{i=1}^{\gamma-1}(\gamma^2 - i) \\ &= 4\gamma^3 - 4\gamma^2 + \gamma\end{aligned}$$

□

Full graph G. Now we describe the additional edges which are part of G but not already in H (see also Figure G.1):

- For $k \in \{1, 2\}$, we connect $G_1^{(k)}, G_2^{(k)}, \ldots, G_{\gamma-1}^{(k)}$ in a chain as follows: every pair (u, v) with $u \in L_i^{(k)}$ and $v \in S_{i+1}^{(k)}$ is an edge (so that the induced subgraph on these two sets of vertices forms a complete bipartite graph).

- At the end of these two chains, we connect all the gadgets $G_\gamma^{(k,j)}$ as follows: every pair (u, v) with $u \in L_{\gamma-1}^{(k)}$ and $v \in S_\gamma^{(k,j)}$ for some j, is an edge.
- Finally we connect these two chains using $G_0 = (S_0, L_0, E_0)$ as follows: every pair (u, v) with $u \in L_0$ and $v \in S_1^{(1)}$ is an edge; and every pair (u, v) with $u \in S_0$ and $v \in S_1^{(2)}$ is an edge.

We note that G is bipartite and all above edges have degree exactly $\deg_H(u) + \deg_H(v) = \beta - 1$, so indeed H is a weighted EDCS of G.

Claim G.5.6. $\mu(G) \geq 4\gamma^3 - 3\gamma^2 + \gamma$.

Proof. We argue that a matching of this size exists in G. In fact the only edges of H we will use as part of this matching are those in the gadgets $G_\gamma^{(k,j)}$.

- We pick a matching between $L_i^{(k)}$ and $S_{i+1}^{(k)}$ of size $|L_i^{(k)}| = \gamma - i$ for all $k \in \{1, 2\}$ and $i = 1, 2, \ldots, \gamma - 2$.
- In $G_\gamma^{(k,j)}$ we pick a matching of size $|L_\gamma^{(k,j)}| = \gamma^2 - \gamma$. Note that exactly 2γ vertices in $S_\gamma^{(k,j)}$ are left unmatched.
- Denote by $U^{(k)}$ the set of unmatched vertices in $S_\gamma^{(k,1)}, S_\gamma^{(k,2)}, \ldots, S_\gamma^{(k,\gamma)}$, for $k \in \{1, 2\}$. Note that $|U^{(k)}| = 2\gamma^2$ and that $(L_{\gamma-1}^{(k)}, U^{(k)})$ forms a complete bipartite graph, so we pick a matching of size $|L_{\gamma-1}^{(k)}| = \gamma^2 - \gamma + 1$ from there.
- Finally we pick matchings between L_0 and $S_1^{(1)}$ (respectively S_0 and $S_1^{(2)}$) of size $|L_0| = \gamma$ (respectively $|S_0| = \gamma$).

In total we see that the above matching is exactly $|S_0| = \gamma$ larger than in Claim G.5.5, which concludes the proof of the claim. □

Approximation ratio. To conclude the proof of Theorem G.5.4, we see that $\mu(G) - \mu(H) = \gamma^2 \geq \frac{1}{4\gamma}\mu(G)$ whenever $\gamma \geq 1$. Hence H preserves at most a $(1 - \frac{1}{4\gamma}) = (1 - \frac{1}{2\sqrt{2\beta}})$ fraction of the maximum matching of G.

G.6 Black-Box Vertex Deletions

Here we briefly explain how one can convert any incremental $(1 - \varepsilon)$-approximate maximum matching algorithm to also support vertex deletions, if allowing additive εn approximation instead, in a black-box fashion. The reduction is simple and also works in general (non-bipartite) graphs. Hence, as an immediate application, we can get a $(1, \varepsilon n)$-approximate matching algorithm for general graphs undergoing both edge-insertions and vertex-deletions, with amortized $1/\varepsilon^{O(1/\varepsilon)}$ update time, if using the incremental algorithm of [GLSSS19].

Lemma G.6.1. *Suppose $\mathcal{A}$ is an algorithm which maintains a $(1-\varepsilon/2)$-approximate maximum matching for a graph undergoing edge insertions, running in total time T. Then there exists an algorithm which maintains a $(1, \varepsilon n)$-approximate matching, in total time $O(T/\varepsilon)$, on a graph undergoing both edge insertions and vertex deletions.*

Proof. We run $\mathcal{A}$, and whenever we get a vertex deletion we ignore it and keep the vertex in the graph. In the outputted matching from the algorithm we remove any edges incident to deleted vertices. When $\varepsilon n/2$ vertices have been deleted, we actually delete them from the graph and rerun the algorithm from scratch (starting on the empty graph). This will only happen $\frac{2}{\varepsilon}$ times, which is the running time blow-up. At each point, the algorithm maintains a matching of size at least $\mu - (\varepsilon n/2 + \varepsilon n/2) = \mu - \varepsilon n$, since only one edge can be removed per deleted vertex still remaining in the graph. □

G.7 Omitted Proofs

EDCS properties

Lemma G.3.2. *In a β-WEDCS H, the maximum degree is at most β and $|H| \leq \beta\mu^*(G)$.*

Proof. If a vertex u has $\deg_H(u) > \beta$, then any incident edge $(u, v) \in H$ is overfull: $\deg_H(u) + \deg_H(v) > \beta$, leading to a contradiction. Hence the maximum degree is at most β. Now we construct a fractional matching by assigning a weight of $1/\beta$ to every edge in H (so an edge appearing with multiplicity ϕ in H gets weight ϕ/β). Clearly this is a feasible fractional matching of G, since no vertex is overloaded. On the other hand, the size of this fractional matching is $|H|/\beta$, implying that $|H| \leq \beta\mu^*(G)$. □

Lemma G.3.3. *If a multiset of edges H is only ever changed by removing overfull edges and adding underfull edges, then there are at most $\beta^2\mu^*(G)$ such insertions/deletions to H.*

Proof. We use a potential function argument. Define

$$\begin{aligned}\Phi(H) &:= |H|(2\beta - 1) - \sum_{(u,v)\in H} (\deg_H(v) + \deg_H(u)) \\ &= \sum_{(u,v)\in H} (2\beta - 1 - \deg_H(u) - \deg_H(v))\end{aligned}$$

We note that if an edge (u, v) appears multiple times in H, it appears multiple times in the above sum as well. We first note that $\Phi(\emptyset) = 0$ and $\Phi(H) \leq \beta|H| \leq \beta^2\mu^*(G)$, by Lemma G.3.2 and since $(2\beta - 1 - \deg_H(u) - \deg_H(v)) \leq \beta$ when H is a valid EDCS. Now we verify that updates to H increase the potential by at least 1:

- Insertion of an underfull edge $(u, v) \in E$.

  That is $\deg_H(u)+\deg_H(v) \leq \beta-2$ before adding the edge. The term $|H|(2\beta-1)$ will increase by $2\beta - 1$. $\sum_{(u,v)\in H}(\deg_H(v) + \deg_H(u))$ will increase by at most $2\beta - 2$, since one term of value $\deg_H(v) + \deg_H(v) \leq \beta - 2 + 2$ (the $+2$ comes from $\deg_H(v)$ and $\deg_H(u)$ increasing by one when we add (u, v) to H) is added, and at most $\beta - 2$ other terms decrease in value by one.

- Deletion of an overfull edge $(u, v) \in H$.

  That is $\deg_H(u) + \deg_H(v) \geq \beta + 1$ before removing the edge. The term $|H|(2\beta - 1)$ will decrease by $2\beta - 1$. $\sum_{(u,v)\in H}(\deg_H(v) + \deg_H(u))$ will decrease by at least 2β, since one term of value $\deg_H(v)+\deg_H(v) \geq \beta+1-2$ (the -2 comes from $\deg_H(v)$ and $\deg_H(u)$ decreasing when we remove (u, v)) is erased, and at least $\beta + 1$ other terms increase in value by one. □

Lemma G.4.2. *If a multiset of edges H is only ever changed by (i) removing overfull edges, (ii) adding underfull edges, and (iii)* removing all edges incident to a vertex when no edges are underfull or overfull*, then there are at most $3\beta^2 n$ insertions/deletions to H.*

Proof. We continue the potential function argument from the proof of Lemma G.3.3 above. When we delete, from H, all edges incident to some vertex u, we know that we deleted at most β many edges from H (as the degree of this vertex was at most β). For each such incident edge (u, v), we bound how much its deletion could have decreased the potential function. The $|H|(2\beta - 1)$ term decreased by exactly $2\beta - 1$, and the $-\sum_{(u,v)\in H}(\deg_H(u) + \deg_H(v))$ term can only increase. So the total decrease in potential, over all up to β incident edges which were deleted, is at most $2\beta^2 - \beta$.

Since we can only delete up to n vertices in total, and the potential is always bounded by $\beta^2\mu^*(G) \leq \beta^2 n$, it follows that the total increase in the potential function, over the run of the algorithm, is at most $3\beta^2 n - n\beta$ (and thus this many updates to H from insertions/deletions of underfull/overfull edges). In total we deleted at most $n\beta$ edges in H incident to deleted vertices, so the total number of updates to H is thus bounded by $3\beta^2 n - \beta n + \beta n = 3\beta^2 n$. □

Approximate Complimentary Slackness

Lemma G.7.1. *Suppose we have the primal linear program* $\max\{c^T x : Ax \leq b, x \geq 0\}$*, and its dual* $\min\{b^T y : A^T y \geq c, y \geq 0\}$*. We say that feasible primal solution x and dual solution y satisfy (α, γ)-approximate complementary slackness (for $\alpha \leq 1 \leq \gamma$) if: (i) if $x_i = 0$ then $(A^T)_i y \leq \gamma c_i$, and (ii) if $y_j = 0$ then $(A)_j x \geq \alpha b_j$. When this is the case, then $\alpha b^T y \leq \gamma c^T x$ (i.e. x and y are $\frac{\gamma}{\alpha}$-approximate optimal).*

Proof. We see that $\gamma c^T x - \alpha b^T y = x^T(\gamma c - A^T y) + y^T(Ax - \alpha b)$. Now either $x_i = 0$ or $(\gamma c - A^T y)_i \geq 0$; and either $y_j = 0$ or $(Ax - \alpha b)_j \geq 0$. Hence $\gamma c^T x - \alpha b^T y \geq 0$. □

Bibliography

[AAGPS19] Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. "Dynamic set cover: improved algorithms and lower bounds". In: *STOC*. ACM, 2019, pp. 114–125. DOI: 10.1145/3313276.3316376 (cit. on p. 375).

[AB19] Sepehr Assadi and Aaron Bernstein. "Towards a Unified Theory of Sparsification for Matching Problems". In: *SOSA*. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 11:1–11:20. DOI: 10.4230/OASICS.SOSA.2019.11 (cit. on pp. 378, 379, 382).

[AB21] Sepehr Assadi and Soheil Behnezhad. "Beating Two-Thirds For Random-Order Streaming Matching". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 19:1–19:13. DOI: 10.4230/LIPICS.ICALP.2021.19 (cit. on p. 378).

[ABBMS19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. "Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs". In: *SODA*. SIAM, 2019, pp. 1616–1635. DOI: 10.1137/1.9781611975482.98 (cit. on p. 378).

[ACCSW18] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. "Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms". In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 7:1–7:16. DOI: 10.4230/LIPICS.ICALP.2018.7 (cit. on p. 375).

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. "Deterministic Fully Dynamic Approximate Vertex Cover and Fractional Matching in O(1) Amortized Update Time". In: *IPCO*. Vol. 10328. Lecture Notes in Computer Science. Springer, 2017, pp. 86–98. DOI: 10.1007/978-3-319-59250-3_8 (cit. on p. 375).

[BDHSS19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. "Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time". In: *FOCS*. IEEE Computer Society, 2019, pp. 382–405. DOI: 10.1109/FOCS.2019.00032 (cit. on p. 375).

[Beh21] Soheil Behnezhad. "Improved Analysis of EDCS via Gallai-Edmonds Decomposition". In: *CoRR* abs/2110.05746 (2021) (cit. on pp. 378, 384, 389).

[Beh23] Soheil Behnezhad. "Dynamic Algorithms for Maximum Matching Size". In: *SODA*. SIAM, 2023, pp. 129–162. DOI: 10.1137/1.9781611977554.CH6 (cit. on pp. 376, 378).

[Ber20] Aaron Bernstein. "Improved Bounds for Matching in Random-Order Streams". In: *ICALP*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 12:1–12:13. DOI: 10.4230/LIPICS.ICALP.2020.12 (cit. on pp. 378, 379, 382).

[BFH21] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. "A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching". In: *ACM Trans. Algorithms* 17.4 (2021). Announced at SODA'19, 29:1–29:51. DOI: 10.1145/3469833 (cit. on p. 375).

[BGS18] Surender Baswana, Manoj Gupta, and Sandeep Sen. "Fully Dynamic Maximal Matching in O(log n) Update Time (Corrected Version)". In: *SIAM J. Comput.* 47.3 (2018). Announced at FOCS'11, pp. 617–650. DOI: 10.1137/16M1106158 (cit. on p. 375).

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. "Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching". In: *SIAM J. Comput.* 47.3 (2018). Announced at SODA'15, pp. 859–887. DOI: 10.1137/140998925 (cit. on p. 375).

[BHN16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. "New deterministic approximation algorithms for fully dynamic matching". In: *STOC*. ACM, 2016, pp. 398–411. DOI: 10.1145/2897518.2897568 (cit. on p. 375).

[BHN17] 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: *SODA*. SIAM, 2017, pp. 470–489. DOI: 10.1137/1.9781611974782.30 (cit. on p. 375).

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. "Deterministically Maintaining a $(2 + \epsilon)$-Approximate Minimum Vertex Cover in $O(1/\epsilon^2)$ Amortized Update Time". In: *SODA*. SIAM, 2019, pp. 1872–1885. DOI: 10.1137/1.9781611975482.113 (cit. on p. 375).

[BK21] Sayan Bhattacharya and Peter Kiss. "Deterministic Rounding of Dynamic Fractional Matchings". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 27:1–27:14. DOI: 10.4230/LIPICS.ICALP.2021.27 (cit. on p. 375).

[BK22] Soheil Behnezhad and Sanjeev Khanna. "New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS". In: *SODA*. SIAM, 2022, pp. 3529–3566. DOI: 10.1137/1.9781611977073.140 (cit. on pp. 375, 376).

[BKS23a] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Dynamic Algorithms for Packing-Covering LPs via Multiplicative Weight Updates". In: *SODA*. SIAM, 2023, pp. 1–47. DOI: 10.1137/1.9781611977554.CH1 (cit. on pp. 376, 377, 384).

[BKS23b] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Sublinear Algorithms for $(1.5+\epsilon)$-Approximate Matching". In: *STOC*. ACM, 2023, pp. 254–266. DOI: 10.1145/3564246.3585252 (cit. on p. 378).

[BKSW23] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. "Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time". In: *SODA*. SIAM, 2023, pp. 100–128. DOI: 10.1137/1.9781611977554.CH5 (cit. on pp. 376, 378).

[BLM20] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. "Fully Dynamic Matching: Beating 2-Approximation in Δ^{ϵ} Update Time". In: *SODA*. SIAM, 2020, pp. 2492–2508. DOI: 10.1137/1.9781611975994.152 (cit. on p. 376).

[BPS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. "Deterministic Decremental Reachability, SCC, and Shortest Paths via Directed Expanders and Congestion Balancing". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 1123–1134. DOI: 10.1109/FOCS46700.2020.00108 (cit. on p. 376).

[BRR23] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Sublinear Time Algorithms and Complexity of Approximate Maximum Matching". In: *STOC*. ACM, 2023, pp. 267–280. DOI: 10.1145/3564246.3585231 (cit. on pp. 376, 378).

[BS15] Aaron Bernstein and Cliff Stein. "Fully Dynamic Matching in Bipartite Graphs". In: *ICALP (1)*. Vol. 9134. Lecture Notes in Computer Science. Springer, 2015, pp. 167–179. DOI: 10.1007/978-3-662-47672-7_14 (cit. on pp. 376, 378, 379, 381, 382, 385, 387).

[BS16] Aaron Bernstein and Cliff Stein. "Faster Fully Dynamic Matchings with Small Approximation Ratios". In: *SODA*. SIAM, 2016, pp. 692–711. DOI: 10.1137/1.9781611974331.CH50 (cit. on pp. 375, 376, 378, 381).

[CKLPGS22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *FOCS*. IEEE, 2022, pp. 612–623. DOI: 10.1109/FOCS54457.2022.00064 (cit. on p. 375).

[CS18] Moses Charikar and Shay Solomon. "Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier". In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 33:1–33:14. DOI: 10.4230/LIPICS.ICALP.2018.33 (cit. on p. 375).

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (1972), pp. 248–264. DOI: 10.1145/321694.321699 (cit. on p. 375).

[GLSSS19] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. "$(1+\epsilon)$-Approximate Incremental Matching in Constant Deterministic Amortized Time". In: *SODA*. SIAM, 2019, pp. 1886–1898. DOI: 10.1137/1.9781611975482.114 (cit. on pp. 376, 377, 391).

[GP13] Manoj Gupta and Richard Peng. "Fully Dynamic (1+ e)-Approximate Matchings". In: *FOCS*. IEEE Computer Society, 2013, pp. 548–557. DOI: 10.1109/FOCS.2013.65 (cit. on pp. 376, 385, 387).

[GSSU22] 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: *SOSA*. SIAM, 2022, pp. 12–23. DOI: 10.1137/1.9781611977066.2 (cit. on pp. 376, 378).

[Gup14] Manoj Gupta. "Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic Update Time". In: *FSTTCS*. Vol. 29. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 227–239. DOI: 10.4230/LIPICS.FSTTCS.2014.227 (cit. on pp. 376, 377).

[HK73] 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. DOI: 10.1137/0202019 (cit. on pp. 375, 381, 385).

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture". In: *STOC*. ACM, 2015, pp. 21–30. DOI: 10.1145/2746539.2746609 (cit. on p. 375).

[JJST22] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. "Regularized Box-Simplex Games and Dynamic Decremental Bipartite Matching". In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 77:1–77:20. DOI: 10.4230/LIPICS.ICALP.2022.77 (cit. on p. 376).

[Kis22] Peter Kiss. "Deterministic Dynamic Matching in Worst-Case Update Time". In: *ITCS*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 94:1–94:21. DOI: 10.4230/LIPICS.ITCS.2022.94 (cit. on pp. 376, 378, 387).

[Kuh55] Harold W Kuhn. "The hungarian method for the assignment problem." In: *Naval research logistics quarterly* (1955) (cit. on p. 375).

[MS04] Marcin Mucha and Piotr Sankowski. "Maximum Matchings via Gaussian Elimination". In: *FOCS*. IEEE Computer Society, 2004, pp. 248–255. DOI: 10.1109/FOCS.2004.40 (cit. on p. 375).

[NS16] Ofer Neiman and Shay Solomon. "Simple Deterministic Algorithms for Fully Dynamic Maximal Matching". In: *ACM Trans. Algorithms* 12.1 (2016). Announced at STOC'13, 7:1–7:15. DOI: 10.1145/2700206 (cit. on p. 375).

[OR10] Krzysztof Onak and Ronitt Rubinfeld. "Maintaining a large matching and a small vertex cover". In: *STOC*. ACM, 2010, pp. 457–464. DOI: 10.1145/1806689.1806753 (cit. on p. 375).

[PS16] David Peleg and Shay Solomon. "Dynamic $(1 + \epsilon)$-Approximate Matchings: A Density-Sensitive Approach". In: *SODA*. SIAM, 2016, pp. 712–729. DOI: 10.1137/1.9781611974331.CH51 (cit. on p. 375).

[RSW22] Mohammad Roghani, Amin Saberi, and David Wajc. "Beating the Folklore Algorithm for Dynamic Matching". In: *ITCS*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 111:1–111:23 (cit. on p. 376).

[San07] Piotr Sankowski. "Faster dynamic matchings and vertex connectivity". In: *SODA*. SIAM, 2007, pp. 118–126 (cit. on p. 375).

[Sol16] Shay Solomon. "Fully Dynamic Maximal Matching in Constant Update Time". In: *FOCS*. IEEE Computer Society, 2016, pp. 325–334. DOI: 10.1109/FOCS.2016.43 (cit. on p. 375).

[Waj20] David Wajc. "Rounding dynamic matchings against an adaptive adversary". In: *STOC*. ACM, 2020, pp. 194–207. DOI: 10.1145/3357713.3384258 (cit. on pp. 375, 384).

[ZH23] Da Wei Zheng and Monika Henzinger. "Multiplicative Auction Algorithm for Approximate Maximum Weight Bipartite Matching". In: *IPCO*. Vol. 13904. Lecture Notes in Computer Science. Springer, 2023, pp. 453–465. DOI: 10.1007/978-3-031-32726-1_32 (cit. on pp. 377, 378).

Paper H

Simple and Asymptotically Optimal Online Bipartite Edge Coloring

Joakim Blikstad, Ola Svensson,
Radu Vintan, David Wajc

Article published in 2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024. [BSVW24b]
Full version at `https://arxiv.org/abs/2311.04574`.

Abstract

We provide a simple online $\Delta(1+o(1))$-edge-coloring algorithm for bipartite graphs of maximum degree $\Delta = \omega(\log n)$ under adversarial vertex arrivals on one side of the graph. Our algorithm slightly improves the result of (Cohen, Peng and Wajc, FOCS19), which was the first, and currently only, to obtain an asymptotically optimal $\Delta(1 + o(1))$ guarantee for an adversarial arrival model. More importantly, our algorithm provides a new, simpler approach for tackling online edge coloring.

H.1 Introduction

Edge coloring is a classic problem in graph theory and algorithm design: *Given a graph, assign colors to the edges, with no two adjacent edges sharing a color.* Pioneering work by König [Kön16] and later Vizing [Viz64] showed that Δ and $\Delta+1$ colors suffice for bipartite and general graphs of maximum degree Δ, respectively. (At least Δ colors are clearly needed.) Algorithms attaining or approximating these bounds were designed in numerous models of computation, including distributed [PS97; Chr23], parallel [KS87], dynamic [DHZ19; Chr23], and streaming algorithms [CL21; ASZ22; CMZ23; GS23; BS23]. The latter includes several simple (asymptotically optimal) $\Delta(1+o(1))$-edge-coloring streaming algorithms for *random-order* streams [CL21; ASZ22].

In contrast, *online* edge-coloring algorithms (especially for *adversarial order*) and their analyses are somewhat more involved [AMSZ03; BMM12; CPW19; BGW21; SW21; KLSST22; NSW23]. The only truly simple online edge coloring algorithm known is the trivial 2-approximate greedy algorithm, which is optimal only for the low-degree regime $\Delta = O(\log n)$ [BMN92]. More involved algorithms were developed for the high-degree setting. For example, all known algorithms for $(\alpha + o(1))$-approximate adversarial-order online edge coloring with $\alpha < 2$ for $\Delta = \omega(\log n)$ [CPW19; SW21; KLSST22; NSW23] rely on interleaved invocations of online matching subroutines that compute a matching that matches each edge e with probability at least $1/(\alpha\Delta)$.[1] The outer loop using such online matching algorithms, introduced by [CPW19], is not particularly complicated, and can be described and analyzed in about one page (see e.g., [SW21, Section 6]). However, the matching algorithms used within this framework and their analyses are quite non-trivial [CW18; CPW19; SW21; KLSST22; NSW23].

We break from the above template, avoiding this outer loop and subsequent complicated online matching subroutines. Instead, we obtain our results by a sequence of *offline* bipartite matching computations (more precisely, random sampling of matchings). This yields a simple asymptotically-optimal online edge coloring algorithm for the first (and so far only) adversarial arrival model for which positive results are known: one-sided vertex arrivals in bipartite graphs [CPW19]. Specifically, we prove the following.

Theorem H.1.1 (See Theorem H.2.4)**.** *There exists an online edge-coloring algorithm for the one-sided vertex arrival model with the following guarantee. On any n-node, maximum degree Δ bipartite graph, it computes a $(\Delta+q)$-edge-coloring with high probability,*[2] *where $q = O(\Delta^{2/3}\log^{1/3} n)$.*

The above theorem only gives non-trivial guarantees if $\Delta \geq q$ (i.e., when $\Delta = \Omega(\log n)$ is sufficiently large). Indeed, when $\Delta < q$, greedy already provides an edge coloring with $2\Delta - 1 \leq \Delta + q$ colors.

[1] Such matchings can be obtained by sampling a color in an $\alpha\Delta$ coloring, so these problems are basically equivalent.

[2] By *with high probability*, we mean probability of at least $1 - n^{-c}$ for some constant $c > 0$.

Our simple online $\Delta(1+o(1))$-edge-coloring algorithm improves on the $o(1)$ term of the algorithm of [CPW19], which uses $\Delta + O(\Delta^{3/4}\log^{1/4} n)$ colors if $\Delta = \omega(\log n)$. Moreover, our simpler algorithm nearly matches a lower bound of $\Delta + \Omega(\sqrt{\Delta})$ colors established in that prior paper. We leave the question of whether an algorithm (simple or otherwise) matching this lower bound's $o(1)$ terms exists as an open problem.

H.2 Simple yet optimal online bipartite edge coloring

Problem statement. A bipartite graph of maximum degree $\Delta = \omega(\log n)$ is revealed.[3] Initially, only n, Δ and the nodes on the *offline* side are known. At time t, the node w_t on the *online* side is revealed, together with its edges, which must be assigned colors immediately and irrevocably. The objective is to compute a valid edge coloring using as few colors as possible.

Our algorithm. We attempt to provide a valid $(\Delta+q)$-edge-coloring, for $q = o(\Delta)$ to be chosen later. In particular, we will color edges of each offline node u with distinct colors, chosen uniformly at random from $\mathcal{C} := [\Delta + q]$. To also color edges of each online node w_t with distinct colors, we correlate the random choices at different offline nodes as follows.

At each time t we consider a bipartite graph H_t with one side given by the set of neighbors $N_G(w_t)$ of the arriving online node w_t in G, and the other side being the set of colors $\mathcal{C}$. The neighbor $v \in N_G(w_t)$ and color $c \in \mathcal{C}$ are connected by an edge $cv \in H_t$ if and only if v has no edge colored c.[4] To color the edges incident to the arriving node w_t in a valid manner, these edges must be given distinct colors and the color chosen for the edge $\{u, w_t\}$ must not already be used at the offline node u. These requirements correspond exactly to matchings in H_t. We thus attempt to sample a matching M_t in H_t where each edge $\{u, w_t\}$ is assigned a uniformly random available color of neighbor u. This can be achieved by a number of randomized rounding algorithms for the bipartite matching polytope, provided the desired marginal matching probabilities lie in this polytope. Fittingly, the crux of our analysis is show that the latter holds w.h.p. for $q = o(\Delta)$ sufficiently large. For simplicity of analysis, we allow for a low-probability "failure mode" if this condition fails, in which case we still insist on coloring offline nodes with colors uniformly at random, but without necessarily providing a valid edge coloring. Our pseudocode is given in Algorithm H.1.

Observation H.2.1. *By definition, we always have $\sum_c x^t_{cv} = 1$ for a vertex $v \in N_G(w_t)$. And so, if $\sum_v x^t_{cv} \le 1$ for all colors $c \in \mathcal{C}$, then the vector $\vec{x}^t$ is in the bipartite matching polytope (of H_t), and a matching M_t as above can be sampled*

[3] As noted above, if Δ is smaller, the problem is solved optimally by the greedy algorithm.

[4] We use the notation $cv \in H_t$ instead of the more standard but notationally cumbersome $\{c, v\} \in E(H_t)$.

Algorithm H.1: Simple Edge Coloring

foreach *arrival of online node* w_t **do**
  Let H_t be a bipartite graph with node sets $N_G(w_t)$ and $\mathcal{C}$, with $cv \in H_t$ iff v has no edge colored c (yet).
  foreach $c \in \mathcal{C}$ *and* $v \in V$ **do**
    Let $x^t_{cv} \leftarrow \frac{\mathbb{1}[cv \in H_t]}{\Delta - d_t(v) + q}$, for $d_t(v)$ the degree of v by time t.
  if $\sum_v x^t_{cv} \leq 1$ *for each color* $c \in \mathcal{C}$ **then**
    Sample matching M_t in H_t with marginals $\Pr[cv \in M_t] = x^t_{cv}$, and color each edge $\{v, w_t\}$ using the color c that is matched to v in M_t.
  else `/* FAILURE MODE */`
    Color each edge $\{v, w_t\}$ with u.a.r. color $c \in N_{H_t}(v)$.

efficiently (and simply, [GKPS06]). In this case, all edges $\{v, w_t\}$ *incident to* w_t *get colored at time* t *(since* $\sum_c x^t_{cv} = 1$*) and they all receive distinct colors from their endpoints' prior and other current edges (due to the definition of* H_t *and* $\sum_v x^t_{cv} \leq 1$*).*

Analysis overview. We wish to show that the condition $\sum_v x^t_{cv} \leq 1$ for all times t and colors c, necessary to avoid the failure mode and output a valid edge coloring, occurs with high probability. For this, we prove two invariants in Lemma H.2.2: we prove (H.1) a closed form for $\mathbb{E}[x^t_{cv}]$, implying $\mathbb{E}[\sum_v x^t_{cv}] \leq 1 - \Omega(q/\Delta)$. If for all t and c these x^t_{cv} were independent, standard Chernoff bounds would suffice to show that w.h.p., $\sum_v x^t_{cv}$ does not deviate much from its expectation, and in particular is at most one. As these variables may be dependent, we also prove (H.2) negative correlation of the random variables x^t_{cv}, allowing us to apply Chernoff-like bounds to these dependent variables and prove that the desired condition holds w.h.p., in Lemma H.2.3.

Lemma H.2.2. *Let* Z^t_{cv} *be the indicator variable for color* c not being used *by edges of* v *when* w_t *arrives. At any time* t*, the following invariants hold:*

- ***(Marginals)*** *For any* color $c \in \mathcal{C}$ *and* offline *node* v*, we have:*

$$\Pr[Z^t_{cv} = 1] = \frac{\Delta - d_t(v) + q}{\Delta + q}. \tag{H.1}$$

- ***(Negative dependence)*** *For any* color $c \in \mathcal{C}$ *and* offline *nodes* $v_1, \ldots, v_k$*, we have:*

$$\Pr\left[\bigwedge_{i \in [k]} (Z^t_{cv_i} = 1)\right] \leq \prod_{i \in [k]} \Pr[Z^t_{cv_i} = 1]. \tag{H.2}$$

Proof. We prove both invariants by induction on $t \geq 1$. The base case $t = 1$ trivially holds for both. To prove both inductive steps, we first note that $\mathbb{1}[cv \in H_t] = Z^t_{cv} \cdot \mathbb{1}[v \in N_G(w^t)]$. So, the value of the random variable x^t_{cv} conditioned on any history up to time t implying $Z^t_{cv} = 1$ is precisely $\overline{x}^t_{cv} := \frac{\mathbb{1}[v \in N_G(w^t)]}{\Delta - d_t(v) + q}$. In particular, conditioning on any such history, the color c is used for edge $\{v, w_t\}$ with probability precisely $\overline{x}^t_{cv}$ (also in the failure mode, and also if $v \notin N_G(w_t)$).

The first invariant's inductive step then follows from the above observation and the inductive hypothesis,by a routine calculation, as follows:

$$\begin{aligned} \Pr[Z^{t+1}_{cv} = 1] &= (1 - \overline{x}^t_{cv}) \cdot \Pr[Z^t_{cv} = 1] \\ &= \left(1 - \frac{\mathbb{1}[v \in N_G(w^t)]}{\Delta - d_t(v) + q}\right) \cdot \frac{\Delta - d_t(v) + q}{\Delta + q} \\ &= \frac{\Delta - d_{t+1}(v) + q}{\Delta + q}. \end{aligned} \tag{H.3}$$

For the second invariant's inductive step, we claim that for any history $\mathcal{H}$ up to time t that implies $\bigwedge_{i \in [k]} (Z^t_{cv_i} = 1)$, we have that $\Pr\left[\bigwedge_{i \in [k]} (Z^{t+1}_{cv_i} = 1) \,\middle|\, \mathcal{H}\right] \leq \prod_{i \in [k]} (1 - \overline{x}^t_{cv})$. This inequality is clearly an equality for the failure mode, where colors are assigned independently; otherwise, the LHS equals $1 - \sum_{i \in [k]} \overline{x}^t_{cv_i}$, which is upper bounded by the RHS, where this standard inequality follows from the union bound. Therefore, by total probability over histories $\mathcal{H}$ as above and the inductive hypothesis and Equation (H.3), we obtain the claimed statement:

$$\begin{aligned} \Pr\left[\bigwedge_{i \in [k]} (Z^{t+1}_{cv_i} = 1)\right] &= \Pr\left[\bigwedge_{i \in [k]} (Z^{t+1}_{cv_i} = 1) \,\middle|\, \bigwedge_{i \in [k]} (Z^t_{cv_i} = 1)\right] \cdot \Pr\left[\bigwedge_{i \in [k]} (Z^t_{cv_i} = 1)\right] \\ &\leq \prod_{i \in [k]} (1 - \overline{x}^t_{cv}) \cdot \prod_{i \in [k]} \Pr[Z^t_{cv_i} = 1] \\ &= \prod_{i \in [k]} \Pr[Z^{t+1}_{cv_i} = 1]. \end{aligned}$$

□

Using these invariants, we now show that Algorithm H.1 is unlikely to enter the failure mode.

Lemma H.2.3. *If $q = 3\Delta^{2/3} \log^{1/3} n \leq \Delta$, then with high probability, for each time t and color $c \in \mathcal{C}$*

$$\sum_v x^t_{cv} \leq 1.$$

Proof. Fix a time t and color c. Notice that $\mathbb{1}[cv \in H_t] = Z^t_{cv} \cdot \mathbb{1}[v \in N_G(w^t)]$, and hence $x^t_{cv} = \frac{Z^t_{cv}}{\Delta - d_t(v) + q}$ for all $v \in N_G(w_t)$ (and $x^t_{cv} = 0$ for all $v \notin N_G(w_t)$). For all $v \in N_G(w_t)$, define the random variables $Y_v := q \cdot x^t_{cv} = \frac{q}{\Delta - d_t(v) + q} \cdot Z^t_{cv}$. It

suffices to prove that $\sum_v Y_v \le q$ with high probability. This follows from a variant of Chernoff bounds, as follows.

First, by Invariant (H.2), because $Y_v \neq 0$ if and only if $Z^t_{cv} = 1$, we have that:

$$\Pr\left[\bigwedge_v (Y_v \neq 0)\right] \le \prod_v \Pr\left[Y_v \neq 0\right].$$

For such weighted binary variables $Y_v \in \{0, \frac{q}{\Delta - d_t(v) + q}\}$, the above is equivalent to the definition of 1-correlation in the sense of [PS97, Definition 3.1], namely $\mathbb{E}[\prod_{v\in U} Y_v] \le \prod_{v\in U} \mathbb{E}[Y_v]$ for all $U \subseteq N_G(w^t)$. As shown in [PS97], this suffices to upper bound the moment-generating function of $\sum_v Y_v$ and derive strong tail bounds. In particular, by [PS97, Corollary 3.3], since we also have that $Y_v \in [0,1]$ for all v, the following Chernoff bound holds for any $\varepsilon > 0$:

$$\Pr\left[\sum_v Y_v \ge (1+\varepsilon)\cdot \mathbb{E}\left[\sum_v Y_v\right]\right] \le \exp\left(-\frac{\varepsilon^2 \cdot \mathbb{E}\left[\sum_v Y_v\right]}{2+\varepsilon}\right). \tag{H.4}$$

Next, by Invariant (H.1), $\mathbb{E}[Y_v] = \frac{q}{\Delta+q}$ for each node $v \in N_G(w_t)$. Hence, $\mathbb{E}[\sum_v Y_v] = \frac{kq}{\Delta+q}$, where $k := |N_G(w_t)| \le \Delta$. By setting $\varepsilon := \frac{\Delta+q-k}{k}$ in the Chernoff bound (H.4) we obtain:

$$\begin{aligned}
\Pr\left[\sum_{v\in N_G(w^t)} Y_v \ge q\right] &= \Pr\left[\sum_{v\in N_G(w^t)} Y_v \ge \left(1 + \frac{\Delta+q-k}{k}\right)\cdot\frac{kq}{\Delta+q}\right] \\
&\le \exp\left(-\frac{(\Delta+q-k)^2}{k^2}\cdot\frac{kq}{\Delta+q}\cdot\frac{k}{\Delta+q+k}\right) \\
&\le \exp\left(-\frac{q^3}{2\Delta^2+3\Delta q+q^2}\right) \\
&\le \exp\left(-\frac{q^3}{6\Delta^2}\right).
\end{aligned}$$

Above, the second-to-last inequality follows because $\left(-\frac{(\Delta+q-k)^2\cdot q}{(\Delta+q)(\Delta+q+k)}\right)$ is decreasing in $k \le \Delta$, and the last inequality relies on $q \le \Delta$ by the lemma's hypothesis. Thus, for our choice of q,

$$\Pr\left[\sum_v x^t_{cv} \ge 1\right] = \Pr\left[\sum_v Y_v \ge q\right] \le \frac{1}{n^{4.5}} \le \frac{1}{2n^3}.$$

The lemma then follows by union bounding over all n online nodes and at most $2n$ colors. □

Combining Observation H.2.1 and Lemma H.2.3, we obtain our result.

Theorem H.2.4. *Algorithm H.1 with $q = 3\Delta^{2/3}\log^{1/3} n \leq \Delta$ (i.e., if $\Delta \geq 81 \log n$) computes a $(\Delta+q)$-edge-coloring of any n-node, maximum degree Δ bipartite graph with high probability.*

Proof. By Lemma H.2.3, the condition $\sum_v x^t_{cv} \leq 1$ holds for all time t and colors c with high probability, which by Observation H.2.1 results in a valid edge coloring using $\Delta + q$ colors. □

Remark H.2.5. In Section H.3, using standard anti-concentration bounds, we also show that our analysis is tight, i.e., that Algorithm H.1 indeed requires $\Delta + \Omega(\Delta^{2/3}\log^{1/3} n)$ colors to work.

Acknowledgements. This work was supported by the Swiss National Science Foundation project 200021-184656 "Randomness in Problem Instances and Randomized Algorithms" and by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00054. Joakim Blikstad is partially supported by the Swedish Research Council (Reg. No. 2019-05622) and the Google PhD Fellowship Program. David Wajc is supported by a Taub Family Foundation "Leader in Science and Technology" fellowship.

APPENDIX

H.3 Tight example for our algorithm

In the following we show that the bound of $(\Delta + O(\Delta^{2/3}\log^{1/3} n))$ colors is tight for Algorithm H.1, for a wide range of Δ superlogarithmic (and even polynomial) in n.

Lemma H.3.1. *For any constant $r \geq 3$, there exists an infinite family of instances with n nodes and maximum degree $\Delta = \Theta(n^{1/r})$, on which Algorithm H.1 run with $q = \frac{1}{6r^{1/3}} \cdot \Delta^{2/3}\log^{1/3} n$ fails to output a valid edge coloring with constant probability.*

Proof. For all (sufficiently large) integer k, we let $\Delta := k - 1$ and construct an instance graph with $n := \max\{k, \lfloor k^{r-2} \rfloor\} \cdot (\Delta^2 + 1) \leq k^{r-2} \cdot (\Delta^2 + 1) \leq k^r$ many nodes. In the instance, $\Delta = k - 1 = \Theta(n^{1/r})$ is the maximum degree of any node in the instance. We turn to describing this instance.

A *gadget* consists of an online node w_t connected to Δ offline neighbors of degree $\Delta - 1$ (before w_t arrives), each of these belonging to disjoint subgraphs. Hence, for any color c these offline nodes are neighbors of c in H_t independently. Our instance consists of $\max\{k, \lfloor k^{r-2} \rfloor\} \geq k$ disjoint (hence independent) such gadgets, each having $\Delta^2 + 1$ nodes and therefore totaling n nodes.

We now fix the gadget corresponding to some w_t. Since the Δ neighbors v of w_t neighbor c independently in H_t, each with probability $\frac{\Delta - d_t(v) + q}{\Delta + q} = \frac{q+1}{\Delta+q}$ (by Invariant (H.1)), the number of neighbors of c in H_t is distributed as $X = \sum_{v \in N_G(w_t)} Z^t_{cv} \sim \text{Bin}(\Delta, \frac{q+1}{\Delta+q})$. Since all neighbors v of c in H_t have $d_t(v) = \Delta - 1$

and hence $x_{cv} = \frac{1}{q+1} \cdot Z^t_{cv}$, Algorithm H.1 does not enter failure mode if and only if $|X| \leq q+1$. We thus wish to lower bound

$$\Pr[X > q+1] = \Pr[X > (1+q/\Delta) \cdot \mathbb{E}[X]] \geq \Pr[X \geq (1+2q/\Delta) \cdot \mathbb{E}[X]]. \quad \text{(H.5)}$$

Let $\varepsilon := 2q/\Delta$. By [KY15, Lemma 4], for $\varepsilon < 1/2$ such that $\varepsilon^2 \cdot \mathbb{E}[X] \geq 3$ (as we shortly verify is the case here), we have the following asymptotic converse of Chernoff's bound:

$$\Pr\left[X \geq (1+\varepsilon) \cdot \mathbb{E}[X]\right] \geq \exp\left(-9\varepsilon^2 \cdot \mathbb{E}[X]\right). \quad \text{(H.6)}$$

To see that the required conditions for applying this inequality hold, first notice that $\varepsilon = \frac{2q}{\Delta} = O(\sqrt[3]{\log n/\Delta})$, and so $\varepsilon < 1/2$ for sufficiently large k (and hence for sufficietly large $\Delta = \Theta(n^{1/r}) >> \log n$). On the other hand, we have that for large enough k (and hence n):

$$\varepsilon^2 \cdot \mathbb{E}[X] = \frac{(2q)^2}{\Delta^2} \cdot \Delta \cdot \frac{q+1}{\Delta+q} \geq \frac{4q^2}{\Delta^2} \cdot \Delta \cdot \frac{q}{2\Delta} = \frac{2q^3}{\Delta^2} = \frac{1}{108r} \cdot \log n \geq 3.$$

Similarly, using that $n \leq k^r$, we have:

$$\varepsilon^2 \cdot \mathbb{E}[X] = \frac{(2q)^2}{\Delta^2} \cdot \Delta \cdot \frac{q+1}{\Delta+q} \leq \frac{4q^2}{\Delta^2} \cdot \Delta \cdot \frac{2q}{\Delta} = \frac{8q^3}{\Delta^2} = \frac{\log n}{27r} \leq \frac{\log k}{9}. \quad \text{(H.7)}$$

Combining the above, we obtain:

$$\Pr[X > q+1] \overset{\text{(H.5)}}{\geq} \Pr[X \geq (1+\varepsilon)\cdot\mathbb{E}[X]] \overset{\text{(H.6)}}{\geq} \exp\left(-9\varepsilon^2 \cdot \mathbb{E}[X]\right) \overset{\text{(H.7)}}{\geq} \exp\left(-\log k\right) = \frac{1}{k}.$$

Hence, Algorithm H.1 enters failure mode on any fixed gadget with probability at least $\frac{1}{k}$. As the instance consists of $\max\{k, \lfloor k^{r-2} \rfloor\} \geq k$ many independent gadgets, the probability that the algorithm does *not enter* failure mode on *any* of them is upper bounded by a constant, $\left(1 - \frac{1}{k}\right)^k \leq 1/e$, or put otherwise $\Pr[\text{enter failure mode}] \geq 1 - 1/e$.

Now, condition on Algorithm H.1 entering failure mode, and fix some time t and color $c \in \mathcal{C}$ for which $\sum_v x^t_{cv} > 1$ (i.e., this is a witness for the algorithm entering failure mode). Then, by the preceding discussion, at least $q+2$ neighbors v of w_t in G are neighbors of c in H_t, where they all have have degree $\Delta - d_t(v) + 1 = q+1$. Therefore, by the independent coloring in the failure mode, the probability that the algorithm fails in outputting a valid edge coloring since it assigns c to two or more

edges of w_t is at least

$$\begin{aligned}\Pr[\text{fail} \mid \text{enter failure mode}] &\geq \Pr\left[\mathrm{Bin}\left(q+2, \frac{1}{q+1}\right) \geq 2\right] \\ &= 1 - \left(1 - \frac{1}{q+1}\right)^{q+2} - \frac{q+2}{q+1} \cdot \left(1 - \frac{1}{q+1}\right)^{q+1} \\ &= 1 - \left(\frac{q}{q+1} + \frac{q+2}{q+1}\right) \cdot \left(1 - \frac{1}{q+1}\right)^{q+1} \\ &\geq 1 - 2/e.\end{aligned}$$

Consequently, Algorithm H.1 fails with constant probability, at least $(1-1/e)(1-2/e)$, as claimed. □

Bibliography

[AMSZ03] Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. "Switch scheduling via randomized edge coloring". In: *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*. 2003, pp. 502–512 (cit. on p. 401).

[ASZ22] Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. "Simple Streaming Algorithms for Edge Coloring". In: *Proceedings of the 30th Annual European Symposium on Algorithms (ESA)*. 2022 (cit. on p. 401).

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. "Online Edge Coloring Algorithms via the Nibble Method". In: *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2021, pp. 2830–2842 (cit. on p. 401).

[BMM12] Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. "Online Graph Edge-Coloring in the Random-Order Arrival Model." In: *Theory of Computing* 8.1 (2012), pp. 567–595 (cit. on p. 401).

[BMN92] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. "The greedy algorithm is optimal for on-line edge coloring". In: *Information Processing Letters (IPL)* 44.5 (1992), pp. 251–253 (cit. on p. 401).

[BS23] Soheil Behnezhad and Mohammad Saneian. "Streaming Edge Coloring with Asymptotically Optimal Colors". In: *arXiv preprint arXiv:2305.01714* (2023) (cit. on p. 401).

[Chr23] Aleksander Bjørn Grodt Christiansen. "The power of multi-step Vizing chains". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 1013–1026 (cit. on p. 401).

[CL21] Moses Charikar and Paul Liu. "Improved algorithms for edge colouring in the w-streaming model". In: *Proceedings of the 4th Symposium on Simplicity in Algorithms (SOSA)*. 2021, pp. 181–183 (cit. on p. 401).

[CMZ23] Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. "Streaming Edge Coloring with Subquadratic Palette Size". In: *arXiv preprint arXiv:2305.07090* (2023) (cit. on p. 401).

[CPW19] Ilan Reuven Cohen, Binghui Peng, and David Wajc. "Tight Bounds for Online Edge Coloring". In: *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 1–25 (cit. on pp. 401, 402).

[CW18] Ilan Reuven Cohen and David Wajc. "Randomized Online Matching in Regular Graphs". In: *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2018, pp. 960–979 (cit. on p. 401).

[DHZ19] Ran Duan, Haoqing He, and Tianyi Zhang. "Dynamic edge coloring with improved approximation". In: *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2019, pp. 1937–1945 (cit. on p. 401).

[GKPS06] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. "Dependent rounding and its applications to approximation algorithms". In: *Journal of the ACM (JACM)* 53.3 (2006), pp. 324–360 (cit. on p. 403).

[GS23] Prantar Ghosh and Manuel Stoeckl. "Low-memory algorithms for online and w-streaming edge coloring". In: *arXiv preprint arXiv:2304.12285* (2023) (cit. on p. 401).

[KLSST22] Janardhan Kulkarni, Yang P Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. "Online Edge Coloring via Tree Recurrences and Correlation Decay". In: *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*. 2022, pp. 2958–2977 (cit. on p. 401).

[Kön16] Dénes König. "Über graphen und ihre anwendung auf determinantentheorie und mengenlehre". In: *Mathematische Annalen* 77.4 (1916), pp. 453–465 (cit. on p. 401).

[KS87] Howard J. Karloff and David B. Shmoys. "Efficient parallel algorithms for edge coloring problems". In: *J. Algorithms* 8.1 (1987), pp. 39–52 (cit. on p. 401).

[KY15] Philip Klein and Neal E Young. "On the Number of Iterations for Dantzig–Wolfe Optimization and Packing-Covering Approximation Algorithms". In: *SIAM Journal on Computing (SICOMP)* 44.4 (2015), pp. 1154–1172 (cit. on p. 407).

[NSW23] Joseph (Seffi) Naor, Aravind Srinivasan, and David Wajc. "Online Dependent Rounding Schemes". In: *arXiv preprint arXiv:2301.08680* (2023) (cit. on p. 401).

[PS97] Alessandro Panconesi and Aravind Srinivasan. "Randomized Distributed Edge Coloring via an Extension of the Chernoff–Hoeffding Bounds". In: *SIAM Journal on Computing (SICOMP)* 26.2 (1997), pp. 350–368 (cit. on pp. 401, 405).

[SW21] Amin Saberi and David Wajc. "The Greedy Algorithm is *not* Optimal for On-Line Edge Coloring". In: *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP)*. 2021, 109:1–109:18 (cit. on p. 401).

[Viz64] Vadim G Vizing. "On an estimate of the chromatic class of a p-graph". In: *Diskret analiz* 3 (1964), pp. 25–30 (cit. on p. 401).

Paper I

Online Edge Coloring is (Nearly) as Easy as Offline

Joakim Blikstad, Ola Svensson,
Radu Vintan, David Wajc

Article published in Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024. [BSVW24a]
Full version at `https://arxiv.org/abs/2402.18339`.

Abstract

The classic theorem of Vizing (Diskret. Analiz.'64) asserts that any graph of maximum degree Δ can be edge colored (offline) using no more than $\Delta + 1$ colors (with Δ being a trivial lower bound). In the *online* setting, Bar-Noy, Motwani and Naor (IPL'92) conjectured that a $(1 + o(1))\Delta$-edge-coloring can be computed online in n-vertex graphs of maximum degree $\Delta = \omega(\log n)$. Numerous algorithms made progress on this question, using a higher number of colors or assuming restricted arrival models, such as random-order edge arrivals or vertex arrivals (e.g., AGKM FOCS'03, BMM SODA'10, CPW FOCS'19, BGW SODA'21, KLSST STOC'22). In this work, we resolve this longstanding conjecture in the affirmative in the most general setting of adversarial edge arrivals. We further generalize this result to obtain online counterparts of the *list* edge coloring result of Kahn (J. Comb. Theory. A'96) and of the recent "local" edge coloring result of Christiansen (STOC'23).

I.1 Introduction

Online edge coloring is one of the first problems studied through the lens of competitive analysis [BMN92]. In this problem, a graph is revealed piece by piece (either edge-by-edge or vertex-by-vertex). An algorithm must assign colors to edges upon their arrival irrevocably so that no two adjacent edges are assigned the same color. The algorithm's objective is to use few colors in any graph of maximum degree Δ, close to the (offline) optimal Δ or $\Delta+1$ guaranteed by Vizing's Theorem [Viz64].

A trivial greedy online algorithm that assigns *any* available color to each edge upon arrival succeeds while using a palette of $2\Delta-1$ colors. As shown over three decades ago, no algorithm does better on low-degree n-vertex graphs with sufficiently small maximum degree $\Delta = O(\log n)$ [BMN92]. While our understanding of online edge coloring is thus complete on *low-degree graphs*, the dynamics are considerably more complex and interesting when Δ surpasses $\omega(\log n)$. The authors in [BMN92] even conjectured that barring the low-degree case, edge coloring can be performed online while nearly matching the guarantees of offline methods.

Conjecture I.1.1 ([BMN92])**.** *There exists an online edge-coloring algorithm for n-vertex graphs that colors the edges of the graph online using $(1+o(1))\Delta$ colors, assuming known maximum degree $\Delta = \omega(\log n)$.*[1]

Progress towards resolving Conjecture I.1.1 was obtained for restricted settings, including random-order edge arrivals [AMSZ03; BMM12; BGW21; KLSST22] and vertex arrivals [CPW19; SW21; BSVW24]. In the most general setting, i.e., under adversarial edge arrivals, [KLSST22] recently provided the first algorithm outperforming the trivial algorithm, showing that an $(\frac{e}{e-1}+o(1))\Delta$-edge-coloring is achievable.

Most prior results for online edge coloring under adversarial arrivals [CPW19; SW21; KLSST22] were attained via the following tight connection between online edge coloring and online matching.[2] Given an $\alpha\Delta$-edge-coloring algorithm, it is easy, by sampling a color, to obtain an online matching algorithm that matches each edge with probability at least $1/(\alpha\Delta)$. In contrast, [CPW19] provided an (asymptotically) optimal reduction in the opposite direction, from online $(\alpha+o(1))\Delta$-edge-coloring algorithms when $\Delta = \omega(\log n)$ to online matching algorithms that match each edge with probability at least $1/(\alpha\Delta)$. (See Lemma I.2.1.) A positive resolution of Conjecture I.1.1 therefore requires—and indeed, is equivalent to—designing an online matching algorithm that matches each edge with probability at least $1/((1+o(1))\Delta)$.

Our Main Result In this work, we resolve Conjecture I.1.1:

[1] Knowledge of Δ is necessary to even use fewer than $\frac{e}{e-1}\Delta \approx 1.582\Delta$ colors [CPW19].

[2] Online matching algorithms must decide for each arriving edge whether to irrevocably add it to their output matching.

Theorem I.1.2 (See exact bounds in Theorem I.4.11)**.** *There exists an online algorithm that, on n-vertex graphs with known maximum degree $\Delta = \omega(\log n)$, outputs a $(1+o(1))\Delta$-edge-coloring with high probability.*

Via the aforementioned reduction, we obtain the above from our following key technical contribution.

Theorem I.1.3. *There exists an online matching algorithm that on graphs with known maximum degree Δ, outputs a random matching M satisfying*

$$\Pr[e \in M] \geq \frac{1}{\Delta + \Theta(\Delta^{3/4}\log^{1/2}\Delta)} = \frac{1}{(1+o(1))\cdot\Delta} \qquad \forall e \in E.$$

We note that the above matching probability of $1/(\Delta+q)$ for $q = \Theta(\Delta^{3/4}\log^{1/2}\Delta)$ approaches a (lower order) lower bound of $q = \Omega(\sqrt{\Delta})$ implied by the competitiveness lower bound of $1 - \Omega(1/\sqrt{\Delta})$ for online matching in regular graphs due to [CW18].

Before explaining further implications of our results and techniques, we briefly discuss our approach for Theorem I.1.3 and its main differences compared to prior work.

Techniques overview For intuition, consider a simple "algorithm" that, by its very design, appears to match every edge with a probability of $1/(\Delta+q)$:

> When an edge $e_t = (u, v)$ arrives and connects two unmatched vertices, match it with probability
>
> $$P(e_t) = \frac{1}{\Delta+q} \cdot \frac{1}{\Pr[u, v \text{ both unmatched until time } t]}.$$

However, the caveat, and the reason for the quotation marks around "algorithm", is that this process is viable only if $P(e_t)$ constitutes a probability. This raises the question: how large must q be to ensure that $P(e_t) \leq 1$ for all edges e_t? Suppose we naively assume that the events "u is unmatched until time t" and "v is unmatched until time t" are independent. In that case, straightforward calculations show that $q = O(\sqrt{\Delta})$ would suffice for well-defined probabilities. However, the assumption of independence rarely holds outside of simplistic graphs like trees, and so the aforementioned events may exhibit complex and problematic correlations. Such correlations present the central challenge in establishing tight bounds for general edge arrivals.

Previous studies addressed the above challenge by circumscribing and managing these correlations. For instance, in more constrained arrival models, [CW18; CPW19] used a variant of this approach; they rely heavily on one-sided vertex arrivals in bipartite graphs to choose an edge to match in a correlated way upon each vertex arrival, while creating useful negative correlation allowing for Chernoff bounds beneficial for future matching choices. In contrast, the only known method for

general edge arrivals was given by [KLSST22]: they subsample locally tree-like graphs and employ sophisticated correlation decay techniques to approximate the independent scenario, albeit at the expense of only being able to match each edge with a probability of at least $1/(\alpha\Delta)$ for $\alpha := e/(e-1) + o(1)$. Unfortunately, the ratio of $e/(e-1)$ appears to be an intrinsic barrier for this approach.

Our approach deviates from the one guiding [CW18; CPW19; KLSST22], by allowing for correlations instead of controlling and taming them. Crucially, we present a different but still simple algorithm, with a subtle difference, which we describe informally here (see detailed exposition in Section I.3): instead of obtaining the probability $P(e_t)$ by scaling $1/(\Delta + q)$ by $1/\Pr[u, v \text{ both unmatched until time } t]$, our scaling factor depends upon the algorithm's actual execution path (sequence of random decisions) so far. The modified algorithm allows us to analyze the scaling factor for an edge as a martingale process. While there may still be correlations, we show that this martingale has (i) small step size and (ii) bounded observed variance. These properties allow for strong Chernoff-type concentration bounds, specifically through Freedman's inequality (Lemma I.2.3), which is pivotal to our analysis. The change of viewpoint is crucial for achieving our result and leads to a simple and concise algorithm and analysis. The results and techniques also extend to more general settings, as we explain next.

Secondary Results and Extensions In Section I.5 we combine Theorem I.1.3 with a new extension of the above-mentioned reduction, from which we obtain online counterparts to two (offline) generalizations of Vizing's theorem, concerning both "local" and list edge coloring. For some background, a *list edge coloring* of a graph is a proper coloring of the edges, assigning each edge a color from an edge-specific palette. The *list chromatic number* of a graph, also introduced by Vizing [Viz76], is the least number of colors needed for each edge to guarantee that a proper list edge coloring exists. A seminal result of Kahn [Kah96] shows that the list chromatic number is asymptotically equal to Δ. Another, "local", generalization of Vizing's Theorem was recently obtained by Christiansen [Chr23], who showed that any graph's edges can be properly colored (offline) with each edge (u, v) assigned a color in the set $\{1, 2, \ldots, 1 + \max(\deg(u), \deg(v))\}$. In this work, using Theorem I.1.3 and extensions of the aforementioned reduction, we show that results of the same flavor as [Kah96] and [Chr23] can be obtained by *online algorithms*.

Theorem I.1.4 (See exact bounds in Theorem I.5.1)**.** *There exists an online algorithm that computes an edge coloring which, with high probability, assigns each edge e a color from its list $L(e)$ (revealed online, with edge e), provided each list has sufficiently large size $(1 + o(1))\Delta$ and that $\Delta = \omega(\log n)$.*

Theorem I.1.5 (See exact bounds in Theorem I.5.2)**.** *There exists an online algorithm that computes an edge coloring assigning each edge $e = (u, v)$ a color from the set $\{1, 2, \ldots, d_{max}(e) \cdot (1 + o(1))\}$ with high probability, where $d_{max}(e) := \max\{\deg(u), \deg(v)\}$, provided that $d_{max}(e) = \omega(\log n)$.*

Finally, in Section I.6, we show that our algorithmic and analytic approach underlying Theorem I.1.3 allows us more generally to round fractional matchings online. Here, an *α-approximate online rounding algorithm* for fractional matchings is revealed (online) an assignment of non-negative values to the edges, $x : E \to \mathbb{R}_{\geq 0}$, with value x_e revealed upon arrival of edge e, so that the total assigned value to the incident edges to each vertex is at most one, $\sum_{e \ni v} x_e \leq 1$. The algorithm's objective is to output online a randomized matching M that matches each edge e with probability at least x_e/α. For *one-sided vertex arrivals* in *bipartite* graphs, it is known that the optimal α is in the range $(1.207, 1.534)$ [NSW23], while if the matching is "sufficiently spread out", $\max_e x_e \leq o(1)$, then $\alpha = 1 + o(1)$ is possible [Waj20, Chapter 5]. We generalize the latter result to the more challenging *edge arrivals* setting in *general* graphs.

Theorem I.1.6 (See exact bounds in Theorem I.6.1)**.** *There exists an online $(1 + o(1))$-approximate rounding algorithm for online matching $\vec{x}$ under adversarial edge arrivals, subject to the promise that* $\max_e x_e \leq o(1)$.

We note that Theorem I.1.3 is the special case of Theorem I.1.6 applied to the fractional matching assigning values $1/\Delta = o(1)$ to each edge. While we focus on this special case in the paper body for ease of exposition, we believe that our more general rounding algorithm is of independent interest and has broader applicability. Illustrating this, in Section I.6.1, we combine our rounding algorithm with a rounding framework and algorithm for fractional edge coloring of [CPW19] to obtain the first online edge coloring algorithm beating the naive greedy algorithm for online edge coloring under vertex arrivals with *unknown* maximum degree $\Delta = \omega(\log n)$; specifically, we show (details in Theorem I.6.11) that $(1.777 + o(1))\Delta$-edge-colorings are attainable in this setting, approaching the lower bound of 1.606Δ proved by [CPW19].

I.1.1 Related Work

Since edge coloring is the problem of decomposing a graph into few matchings, it is natural to relate this problem to online matching.

The study of online matching was initiated by Karp, Vazirani and Vazirani [KVV90], whose main result was a positive one: they presented an optimal algorithm under one-sided vertex arrivals in bipartite graphs, showing in particular that the greedy algorithm's competitive ratio is suboptimal for this problem. Similar positive results were later obtained for several generalizations, including weighted matching [AGKM11; FHTZ20; BC21; GHHNYZ21], budgeted allocation (a.k.a AdWords) [MSVV07; HZZ20], and fully-online matching [HKTWZZ20; HTWZ20]. However, in the most general setting, i.e., under edge arrivals, the competitive ratio of the trivial greedy algorithm is optimal [GKMSW19].

The study of online edge coloring was initiated by Bar-Noy, Motwani and Naor [BMN92], who presented a negative result: they showed that the greedy algorithm is optimal, at least for low-degree graphs. Positive results were later obtained

under random-order arrivals [AMSZ03; BMM12], culminating in a resolution of Conjecture I.1.1 for such arrivals, using the nibble method [BGW21]. For adversarial arrivals, [CPW19; BSVW24] show that in bipartite graphs with one-sided vertex arrivals, the same conjecture holds. This was followed by progress in general graphs, under vertex arrivals [SW21], and edge arrivals [KLSST22], though using more than the hoped-for $(1+o(1))\Delta$ many colors. We obtain this bound in this work. Thus, we show that not only is the greedy algorithm suboptimal for online edge coloring, but in fact in the most general edge arrival setting, the online problem is asymptotically no harder than its offline counterpart.

Follow-up work. Given the randomized results obtained in this work, a natural remaining open question is whether *deterministic* online edge coloring algorithms can achieve better competitive ratios than the greedy algorithm. Dudeja, Goswani and Saks [DGS24] have since shown that for any constant $\epsilon > 0$ and $M > 0$, there exists a deterministic online edge-coloring algorithm using $(1+\epsilon)\Delta$ colors for graphs of sufficiently large linear-in-n degree $\Delta \geq n/M$. In another work, [BSVW25] we provided a deterministic $(\frac{e}{e-1} + o(1))\Delta$-edge-coloring algorithm under one-sided arrivals for bipartite graphs of any super-logarithmic maximum degree $\Delta = \omega(\log n)$, matching the degree bound needed of the deterministic lower bound of [BMN92]. It remains unknown whether both the number of colors and degree bound of the above two papers can be achieved simultaneously, in effect de-randomizing the current paper.

I.2 Preliminaries

Notation. As standard, we denote by $N(v)$ and $\delta(v)$ the neighborhood and edge sets of v, respectively, and denote the number of vertices and edges of G by $n := |V|$ and $m := |E|$. We also denote by $\deg_H(v)$ the degree of vertex v in (sub)graph H, and use the shorthand $\deg(v) := \deg_G(v)$. We say an event happens *with high probability in a parameter* k if it happens with probability at least $1 - k^{-c}$ for a constant $c > 0$.

Problem definition and notation In the online problems studied in this paper, the input is an undirected simple graph $G := (V, E)$ with known maximum degree Δ. Its edges arrive one at a time, with edge $e_t \in E$ arriving at time t. An *online edge coloring algorithm* must color each edge e_t upon arrival with a color distinct from its adjacent edges. Similarly, an *online matching algorithm* must decide whether to match e_t upon arrival, if none of its endpoints are matched. For both problems, we consider randomized algorithms and assume that the input is generated by an oblivious adversary, which fixes the input graph and edges' arrival order before the algorithm receives any input.[3] The objective of edge coloring algorithms is to output

[3] By standard reductions [BBKTW94], a result quantitatively similar to our main result against an *adaptive* adversary would be equivalent to the task of finding a *deterministic* algorithm.

a coloring using as few colors as possible, close to the offline optimal Δ or $\Delta+1$ colors [Viz64]. The objective of online matching algorithms is traditionally to output a large matching. However, due to the reduction mentioned in the introduction, and restated more formally below, our interest will be in online matching algorithms that match each edge with high probability, close to $1/\Delta$.

Lemma I.2.1 (Reduction ([CPW19; SW21]))**.** *Let $\mathcal{A}$ be an online matching algorithm that, on any graph of maximum degree $\Delta = \omega(\log n)$, matches each edge with probability at least $1/(\alpha \cdot \Delta)$, for $\alpha \geq 1$. Then, there exists an online edge coloring algorithm $\mathcal{A}'$ that on any graph with maximum degree $\Delta = \omega(\log n)$ outputs an edge coloring with $(\alpha + O((\log n/\Delta)^{1/4})) \cdot \Delta$ colors with high probability in n.*

In Section I.5, we generalize the above lemma, and use this generalization to obtain results for online *list* edge coloring (each edge has a possibly distinct palette) and for online *local* edge coloring (each edge e should be colored with a color of index not much higher than $\max_{v \in e} \deg(v)$). In particular, the appendix implies (see Lemma I.5.15) that one can reduce the slack above to $(\alpha + O((\log n/\Delta)^{1/3})) \cdot \Delta$ colors.

Martingales A crucial ingredient in the analysis of our algorithms is the use of martingales.

Definition I.2.2 (Martingale)**.** A sequence of random variables $Y_0, \ldots, Y_m$ is a *martingale with respect to* another sequence of random variables $X_1, \ldots, X_m$ if the following conditions hold:

- Y_k is a function of $X_1, \ldots, X_k$ for all $k \geq 1$.
- $\mathbb{E}[|Y_k|] < \infty$ for all $k \geq 0$.
- $\mathbb{E}[Y_k \mid X_1, \ldots, X_{k-1}] = Y_{k-1}$ for all $k \geq 1$.

The technical advantage of using martingales in our analysis is their amenability to specialized concentration inequalities which, unlike Chernoff-Hoeffding type bounds, do not require independence (or negative correlation) between the involved random variables. In particular, we will use a classic theorem due to Freedman providing a Chernoff-type bound only depending on the *step size* and on the *observed variance* of the martingale, with the latter defined as follows. For any possible outcomes $(x_1, \ldots, x_{m-1})$ of the random variables $(X_1, \ldots, X_{m-1})$, let:

$$W_m(x_1, \ldots, x_{m-1}) := \sum_{i=1}^{m} \mathbb{E}[(Y_i - Y_{i-1})^2 \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}]$$

be the *observed variance* encountered by the martingale on the particular sample path $x_1, \ldots, x_{m-1}$ it took. To simplify notation, we usually assume that $x_1, \ldots, x_{m-1}$

are chosen arbitrarily and write:

$$W_m := \sum_{i=1}^{m} \mathbb{E}[(Y_i - Y_{i-1})^2 \mid X_1, \ldots, X_{i-1}].$$

Lemma I.2.3 (Freedman's Inequality [Fre75]; see also [BDG16, Theorem 12], [HMRR98, Theorem 3.15])**.** *Let $Y_0, \ldots, Y_m$ be a martingale with respect to the random variables $X_1, \ldots, X_m$. If $|Y_k - Y_{k-1}| \leq A$ for any $k \geq 1$ and $W_m \leq \sigma^2$ always, then for any real $\lambda \geq 0$:*

$$\Pr[|Y_n - Y_0| \geq \lambda] \leq 2 \exp\left(-\frac{\lambda^2}{2(\sigma^2 + A\lambda/3)}\right).$$

We remark that we tailored the inequality to our use, and a more general version holds [Fre75].

I.3 Online Matching Algorithm

In this section, we design an online matching algorithm as guaranteed by Theorem I.1.3.

I.3.1 Our First Matching Algorithm

We first describe our key modification of the basic algorithm presented in the introduction. Analogous to that algorithm, upon arrival of edge e_t whose endpoints are still unmatched, we match e_t with a "scaled" probability $P(e_t)$. However, and crucially for our analysis, our scaling factor will depend on the specific execution (random choices) of the algorithm. To illustrate this modification, refer to Figure I.1, which depicts the neighborhood surrounding e_t. Here, we denote by e_{t_j} the j-th edge connecting a vertex in e_t with a vertex w_j, with these $k = 7$ edges appearing sequentially before e_t, with $t_1 < t_2 < \cdots < t_k < t$. Suppose now that we fixed the randomness associated with all edges other than $\{e_t, e_{t_1,}, e_{t_2}, \ldots, e_{t_k}\}$, and conditioned on this event, which we refer to as R, let $P_R(e_{t_j})$ represent the probability that the algorithm will add the edge e_{t_j} to the matching, assuming that none of the

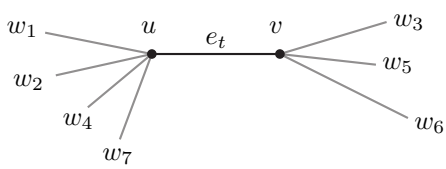


Figure I.1: An example of the neighborhood of $e_t = (u, v)$ with $k = 7$.

preceding edges $e_{t_1}, e_{t_2}, \ldots, e_{t_{j-1}}$ have been matched. Consequently, we have:

$$\Pr[u, v \text{ both unmatched until time } t \mid R] = \prod_{j=1}^{k} (1 - P_R(e_{t_j})),$$

and overall, by total probability,

$$\Pr[u, v \text{ both unmatched until time } t] = \sum_R \Pr[R] \cdot \prod_{j=1}^{k} (1 - P_R(e_{t_j})) \,.$$

The primary complication with the initial algorithm in the introduction is the intricate correlation within the joint distribution of $P_R(e_{i_1}), \ldots, P_R(e_{i_k})$ as functions of the randomness R of edges outside the direct neighborhood of e_t. This correlation complicates both the computational aspect of determining the probability that vertices u and v are unmatched until time t, as well as the theoretical analysis of the algorithm's competitive ratio.

Our algorithm overcomes this challenge with a, in hindsight, simple strategy. We utilize a scaling factor conditional on the randomness R, i.e., we scale with respect to the "observed" probabilities, thus ensuring that the resulting online algorithm is both computationally efficient and (as we will see) theoretically tractable to analyze. Specifically, we obtain the following algorithm:

Algorithm I.1: NaturalMatchingAlgorithm

1 When an edge $e_t = (u, v)$ arrives, match it with probability

$$P(e_t) \leftarrow \begin{cases} \frac{1}{\Delta+q} \cdot \frac{1}{\prod_{j=1}^{k}(1-P(e_{t_j}))} & \text{if } u \text{ and } v \text{ are still unmatched,} \\ 0 & \text{otherwise,} \end{cases}$$

where $e_{t_1}, \ldots, e_{t_k}$ are those previously-arrived edges incident to the endpoints of e_t.

Note that the values $P(e_{t_j})$ needed to compute $P(e_t)$ are all defined at time t (and easy to compute), since any such edge e_{t_j} arrived before e_t. Moreover, assuming u and v are unmatched (*free*), these values equal $P_R(e_{t_j})$, where R is the event corresponding to the random bits used in this execution for the edges outside the neighborhood of e_t. Thus, if $P(e_t) \leq 1$ for every edge e_t, this algorithm is well-defined, and attains the right marginals, by total probability over R:

$$\begin{aligned} \Pr[e_t \text{ matched}] &= \sum_R \Pr[R] \cdot P_R(e_t) \cdot \Pr[u, v \text{ both free until time } t \mid R] \\ &= \sum_R \Pr[R] \cdot \frac{1}{\Delta + q} = \frac{1}{\Delta + q} \,. \end{aligned}$$

However, our new natural algorithm is ill-defined, as $P(e_t)$ may exceed 1, even on trees, as the following example illustrates.

Suppose edges in Figure I.2 arrive in a bottom-up, left-to-right order, and no edge before e_t is matched by the algorithm (this is a very low-probability event). Then $P(e_{t_1}) = \frac{1}{\Delta+q} \cdot \frac{1}{\prod_{i=1}^{\Delta-2}(1-P(e_i))}$, where $e_1, \ldots, e_{\Delta-2}$ are the edges below e_{t_1}. But the term $\prod_i(1-P(e_i))$ is the probability that none of the edges e_i are matched by the algorithm, which is exactly $1 - \frac{\Delta-2}{\Delta+q} = \frac{q+2}{\Delta+q}$, since each e_i in this example is matched with probability exactly $\frac{1}{\Delta+q}$ and these are disjoint events. Hence $P(e_{t_1}) = \frac{1}{\Delta+q} \cdot \frac{\Delta+q}{q+2} = \frac{1}{q+2}$. We can calculate $P(e_{t_2})$ in the same way, remembering to also scale up by $\frac{1}{1-P(e_{t_1})}$, and we get $P(e_{t_2}) = \frac{1}{q+2} \cdot \frac{1}{1-P(e_{t_1})} = \frac{1}{q+1}$. Continuing in this fashion, $P(e_{t_i}) = \frac{1}{q+3-i}$ for any $i = 1, \ldots, q+1$ (importantly $P(e_{t_i}) < 1$, so indeed with some non-zero probability the algorithm will not match any of them). Finally, by a similar calculation, $P(e_t) = \frac{1}{q+1} \cdot \frac{1}{\prod_{i=1}^{q+1}(1-P(e_{t_i}))} = \frac{q+2}{q+1} > 1$, making the algorithm undefined. However, in *most* runs of the algorithm, many bottom-level edges are matched, and so $P(e_{t_i}) = 0$ for many i, and $P(e_t)$ is much smaller.

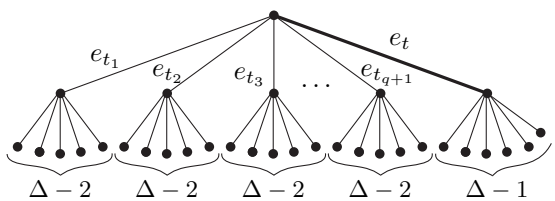


Figure I.2: Example where Algorithm I.1 might be undefined.

In the above example, the event that $P(e_t)$ in this case exceeds one hinges on low-probability events. This therefore does not rule out matching probabilities of, say, $1/(\Delta+q) - 1/\Delta^3 \geq 1/(\Delta + O(q))$, so long as we avoid the use of $P(e_t)$ as probabilities when $P(e_t) > 1$. In the next section we present a modification of Algorithm I.1 doing just this, and provide an overview of its analysis.

I.3.2 The Analysis-Friendly Matching Algorithm

As discussed, it is not at all clear whether the random variables $P(e_t)$ in Algorithm I.1, which we interpret as probabilities, even *are* valid probabilities, i.e., whether they are upper bounded by 1. To avoid working with potentially invalid probabilities, we use a slightly different variant of Algorithm I.1, whose pseudocode is given by Algorithm I.2. This variant not only addresses the ill-defined probability concern, but also introduces a more precise notation, facilitating our analysis. However, this increased precision might initially obscure the connection between Algorithm I.2 and the more intuitive Algorithm I.1.

To clarify this relationship, consider the scaling factor $1/\prod_{j=1}^{k}(1-P(e_{t_j}))$ for an edge $e_t = (u, v)$, which can be partitioned based on the edges incident to vertices

u and v. Let $\delta_t(u)$ denote the set of edges incident at u and arriving before e_t. We then define $F_t(u)$ (and similarly $F_t(v)$) as follows:

$$F_t(u) := \prod_{e_{t_j} \in \delta_t(u)} (1 - P(e_{t_j})) \qquad \text{and} \qquad F_t(v) := \prod_{e_{t_j} \in \delta_t(v)} (1 - P(e_{t_j})).$$

Consequently, as G is a simple graph, the probability $P(e_t)$ used in Algorithm I.1 can be reformulated as:

$$P(e_t) = \begin{cases} \frac{1}{\Delta+q} \cdot \frac{1}{F_t(u) \cdot F_t(v)} & \text{if } u \text{ and } v \text{ are still unmatched by time } t, \\ 0 & \text{otherwise.} \end{cases}$$

The above aligns with the pseudo-code in Algorithm I.2. Assuming that $\hat{P}(e_t) = P(e_t)$ (and using that G is a simple graph and hence has no parallel edges), it becomes evident that Algorithm I.2 is equivalent to Algorithm I.1 under the premise that all probabilities $P(e_t)$ are at most one. The critical modification in Algorithm I.2 is the introduction of $\hat{P}$, possibly distinct from P, ensuring that $P(e_t) \leq 1$. This is accomplished by constraining the values of $F_t(v)$, now redefined in terms of $\hat{P}$, i.e., $F_t(v) := \prod_{e_{t_j} \in \delta_t(v)}(1 - \hat{P}(e_{t_j}))$, to not fall below $\frac{q}{4\Delta}$, implying $P(e_t) \leq 1$ for appropriately small q (see Observation I.4.3).

Analysis: Intuition and Overview As in Algorithm I.1, it is not hard to prove that edge e_t is selected with a probability of $1/(\Delta + q)$ if $P(e_t)$ always equals $\hat{P}(e_t)$, as detailed in Section I.4.1. Therefore, the meat of our analysis, in Section I.4.2, focuses on proving that for any edge e_t, the equality $P(e_t) = \hat{P}(e_t)$ holds with high probability in Δ. For intuition why this should be true, note that if each of the $\deg_t(v)$ edges of vertex v by time t is matched with probability $\frac{1}{\Delta+q}$, then the value $F_t(v)$—that intuitively stands for the probability of v being free at time t—should be:

$$F_t(v) = 1 - \frac{\deg_t(v)}{\Delta + q} \geq \frac{q}{\Delta + q} \approx \frac{q}{\Delta}.$$

Above, the inequality follows from $\deg_t(v) \leq \Delta$ and the approximation follows by our choice of $q = o(\Delta)$. Moreover, basic calculations (see Observation I.4.4) imply that $\hat{P}(e_t) = P(e_t)$ if $\min\{F_t(u), F_t(v)\} \geq \frac{q}{3\Delta}$. In other words, $\hat{P}(e_t) \neq P(e_t)$ only if the $F_t(\cdot)$-value has dropped significantly below its "expectation" for one of the endpoints of e_t.

The core of the analysis then boils down to proving concentration bounds that imply, for any vertex v and time t, that $F_t(v) \geq \frac{\Delta}{3q}$ with high probability in Δ. The values $F_t(v)$ are non-increasing as t grows, so it suffices to prove this inequality for the final value $F(v) := F_m(v)$. Let $u_1, \ldots, u_\ell$ be the neighbors of v in the final graph. By simple calculations (Lemma I.4.5), we show that:

$$F(v) \geq 1 - \sum_{i=1}^{\ell} E_i \cdot \frac{1}{\Delta + q} \cdot \frac{1}{F_{t_i}(u_i)}, \tag{I.1}$$

Algorithm I.2: MATCHINGALGORITHM

1 ***Initialization:*** Set $F_1(v) \leftarrow 1$ for every vertex v and $M_1 \leftarrow \emptyset$.

2 *At the arrival of edge $e_t = (u, v)$ at time t:*

- Sample $X_t \sim \text{Uni}[0, 1]$.
- Define

$$P(e_t) = \begin{cases} \frac{1}{\Delta+q} \cdot \frac{1}{F_t(u) \cdot F_t(v)} & \text{if } u \text{ and } v \text{ are unmatched in } M_t, \\ 0 & \text{otherwise.} \end{cases}$$

  and

$$\hat{P}(e_t) = \begin{cases} P(e_t) & \text{if } \min\{F_t(u), F_t(v)\} \cdot (1 - P(e_t)) \geq q/(4\Delta) \\ 0 & \text{otherwise.} \end{cases}$$

- Set
  - $F_{t+1}(u) \leftarrow F_t(u) \cdot (1 - \hat{P}(e_t))$;
  - $F_{t+1}(v) \leftarrow F_t(v) \cdot (1 - \hat{P}(e_t))$;
  - $M_{t+1} \leftarrow \begin{cases} M_t \cup \{e_t\} & \text{if } X_t < \hat{P}(e_t), \\ M_t & \text{otherwise.} \end{cases}$

where the binary random variables E_i are non-zero if and only if the neighbor u_i of v is unmatched by the time t_i at which the edge (v, u_i) arrives. If we denote by $S := \{u_i \in N(v) \mid u_i \text{ not matched until time } t_i\}$, then $E_i := \mathbb{1}[u_i \in S]$.

The expectation of the sum $Y := \sum_{i=1}^{\ell} E_i \cdot \frac{1}{\Delta+q} \cdot \frac{1}{F_{t_i}(u_i)}$ can be shown to be at most $\frac{\Delta}{\Delta+q}$. If the variables E_i and $F_{t_i}(u_i)$ were independent, one could now use Chernoff-Hoeffding type bounds to conclude that $Y \leq \frac{\Delta}{\Delta+q/2}$ with high probability in Δ, proving $F(v) \geq 1 - Y_m \geq \frac{q}{3\Delta}$ in the process (see Lemma I.4.5). However, in general, the events of different neighbors u_i of v being matched when (v, u_i) arrives are not independent, and so the variables $E_i, F_{t_i}(u_i)$ are correlated, making such an approach not applicable. We overcome this by interpreting the right-hand side of (I.1) as the final state of a martingale.

Concretely, our main idea is to parameterize the set S, the random variables E_i and the sum Y over time. Assuming that the input stops at time step $t \leq m$, one can naturally define the analogues of S, E_i, and Y up to time t by $S_t := \{u_i \in$

$N(v) \mid u_i$ not matched until time $\min\{t, t_i\}\}$, $E_{ti} := \mathbb{1}[u_i \in S_t]$ and:

$$Y_t := \sum_i^{\ell} E_{ti} \cdot \frac{1}{\Delta + q} \cdot \frac{1}{F_{\min\{t,t_i\}}(u_i)}.$$

So, at each time step, we either change the value of a term or drop a term. With this notation, we trivially have that $S = S_m$, $E_i = E_{mi}$ and $Y = Y_m$. The advantage of this representation is that $Y_0, \ldots, Y_m$ turns out to be a martingale with respect to the random variables $X_1, \ldots, X_m$ sampled by Algorithm I.2. As $Y_0 \leq \frac{\Delta}{\Delta+q}$, our objective reduces to proving that the following holds with high probability in Δ:

$$|Y_m - Y_0| \leq \frac{\Delta}{\Delta + q/2} - \frac{\Delta}{\Delta + q}.$$

As the martingale $Y_0, \ldots, Y_m$ takes Δ^2 non-trivial steps (based on the two-hop neighborhood), it is not enough to use the maximum step size and the number of steps to argue about concentration as done in, e.g., Azuma's inequality. However, we can bound the maximum step size and the observed variance, which is sufficient for applying Freedman's inequality (Lemma I.2.3), yielding our desired result. In the next section we substantiate the above intuition, and analyze Algorithm I.2.

I.4 Analysis of the Online Matching Algorithm

In this section we present the formal analysis of Algorithm I.2, and prove that it matches each edge e_t with probability at least $1/(\Delta + O(q))$, with q to be chosen shortly. Our analysis is divided into two parts.

In the first part (Section I.4.1), we prove that if $\hat{P}(e_t) = P(e_t)$, i.e., the values $F_t(v)$ for both $v \in e_t$ are large enough, then we match e_t with probability at least $1/(\Delta + q)$ (Lemma I.4.2).

In the second part (Section I.4.2), we remove the assumption that $P(e_t) = \hat{P}(e_t)$ and prove that Algorithm I.2 achieves a matching probability of at least $1/(\Delta + 4q)$, by showing that with high probability in Δ, the values $F_t(v)$ are large enough to guarantee $P(e_t) = \hat{P}(e_t)$. To prove the latter high-probability bound, we interpret a sufficient desired lower bound (Lemma I.4.5) as the final state of a martingale, and use Freedman's inequality (Lemma I.2.3) to prove that $F_t(v)$ is likely sufficiently large for our needs.

Choice of q We will use $q := \sqrt{200} \cdot \Delta^{3/4} \ln^{1/2} \Delta$, for reasons that will become clear in the proof of Lemma I.4.10. For the rest of the section we will only make use of the following corollaries of our choice of q:

$$8\sqrt{\Delta} \leq q \leq \Delta/4. \tag{I.2}$$

Note that the upper bound on q not only follows from its choice (for sufficiently large Δ), but we may also assume this bound without loss of generality: if $q > \Delta/4$,

then simply picking a random color used by the $(2\Delta - 1)$ colors of the greedy online coloring algorithm will match each edge with probability $1/(2\Delta - 1)$, greater than our desired $1/(\Delta + 4q)$ matching probability.

I.4.1 A Sufficient Condition for $1/(\Delta + q)$ Matching Probability

We begin with a simple observation, which will facilitate our characterization of random values associated with Algorithm I.2 under various conditionings.

Observation I.4.1. *For any time t, the random variables $F_t(v), P(e_t), \hat{P}(e_t)$ are determined by the current partial input $e_1, \ldots, e_t$ and the current matching M_{t-1}.*

Proof. Since $P(e_t)$ and $\hat{P}(e_t)$ are determined by the values of the variables $F_t(v)$, it suffices to prove the statement only for these latter variables. This follows by induction on t. For the base case, we have $F_1(v) = 1$ for all vertices, which implies the statement trivially. For the inductive step with $t > 1$, note that by construction $F_t(v)$ is determined by the values $\{\hat{P}(e_{t'}) : t' < t \text{ and } e_{t'} = (u', v) \text{ is incident to } v\}$. Any such value $\hat{P}(e_{t'})$ is in turn a function of $F_{t'}(u'), F_{t'}(v)$ and $P(e_{t'})$. By the inductive hypothesis, $F_{t'}(u'), F_{t'}(v)$ are functions of $M_{t'-1}$ and therefore also functions of M_{t-1} (since $t' < t$). Finally, the value $P(e_{t'})$ is determined by $F_{t'}(u'), F_{t'}(v)$ and $M_{t'-1}$, which are again functions of M_{t-1}. □

The following lemma formalizes the intuition discussed in Section I.3; it proves that Algorithm I.2 has the correct behavior for an edge $e_t = (u, v)$ by assuming that the randomness outside the 1-neighborhood $(\delta(u) \cup \delta(v))$ of e_t is fixed and $\hat{P}(e_t) = P(e_t)$.

Lemma I.4.2. *For any edge $e_t = (u, v)$ it holds that*

$$\Pr[X_t < P(e_t)] = \frac{1}{\Delta + q}.$$

Proof. Fix all randomness except for the edges incident to u and v. That is, we fix the outcomes $X_{t'} = x_{t'}$ for all edges $e_{t'} \notin \delta(u) \cup \delta(v)$. Denote this event by $A(\vec{x})$. Let $t_1 < \cdots < t_\ell$ be the arrival times of the edges in $\delta(u) \cup \delta(v)$ before time t. The only randomness left up to the point of e_t's arrival is now given by the random variables $X_{t_1}, \ldots, X_{t_\ell}$, which are *independent* of $A(\vec{x})$. For the selection of e_t to be possible, we need to condition on the event that none of the edges $e_{t_1}, \ldots, e_{t_\ell}$ are taken in the matching. We note that conditioning on $A(\vec{x})$ and $e_{t_1}, \ldots, e_{t_\ell} \notin M_t$ completely determines $M_{t'}$ for all time steps $t' \leq t$. Using Observation I.4.1, we thus have that $\hat{P}(e_{t_1}), \ldots, \hat{P}(e_{t_\ell})$, and $F_t(u), F_t(v)$ are uniquely determined under this conditioning. Let $\hat{p}(e_{t_1}), \ldots, \hat{p}(e_{t_\ell})$, and $f_t(u), f_t(v)$ be the concrete values of

these random variables under this conditioning. We then have:

$$\Pr[\{e_{t_1}, \ldots, e_{t_\ell}\} \cap M_t = \emptyset \mid A(\vec{x})] = \prod_{i=1}^{\ell} \Pr[X_{t_i} > \hat{p}(e_{t_i})] = \prod_{i=1}^{\ell} (1 - \hat{p}(e_{t_i})) = f_t(u) \cdot f_t(v).$$

Above, for the first equality we used the independence of the different X_{t_i}, while for the last equality we partitioned the edges incident to u and v to obtain the factors $f_t(u)$ and $f_t(u)$ respectively. Note that, because the graph is simple and so parallel edges are disallowed, this partitioning is well defined, as none of the edges e_{t_i} connects u to v. If u or v are unmatched before time t, we get: $P(e_t) = \frac{1}{\Delta+q} \cdot \frac{1}{f_t(u) \cdot f_t(v)}$. Since the random variable X_t is independent from M_t, the above then yields the desired equality when conditioning on $A(\vec{x})$:

$$\begin{aligned}\Pr[X_t < P(e_t) \mid A(\vec{x})] &= \Pr[X_t < P(e_t) \text{ and } \{e_{t_1}, \ldots, e_{t_\ell}\} \cap M_t = \emptyset \mid A(\vec{x})] \\ &= \frac{1}{\Delta + q} \cdot \frac{1}{f_t(u) \cdot f_t(v)} \cdot f_t(u) \cdot f_t(v) \\ &= \frac{1}{\Delta + q}.\end{aligned}$$

The lemma now follows by the law of total probability over all possible values of $A(\vec{x})$. □

Note that a consequence of the above lemma is that, if the values of P and $\hat{P}$ were to *always* coincide during the execution of Algorithm I.2, then all edges would be matched with probability $1/(\Delta + q)$. In the following section, we prove that the equality $\hat{P}(e_t) = P(e_t)$ holds with high probability in Δ for any edge e_t, which, by simple calculations, implies a matching probability of $1/(\Delta + O(q))$.

I.4.2 Analysis of Algorithm I.2 in General

In this section we wish to prove that $\hat{P}$ and P coincide for each edge with high probability in Δ. This requires proving that $F_t(v)$ is likely to be high for all times t and vertices v—intuitively that means that the probability that v is unmatched is never too low. We start by observing a trivial lower bound on $F_t(v)$ and upper bound on $P(e)$ that follows directly from the algorithm's definition.

Observation I.4.3. *$F_t(v) \geq q/(4\Delta)$ and $\hat{P}(e_t) \leq P(e_t) \leq 1/4$ for all vertices $v \in V$ and times t.*

Proof. For any fixed v, we prove the first statement by induction on t. For $t = 0$, using (I.2), we have $F_t(v) = 1 \geq q/(4\Delta)$. Now, assume that $F_t(v) \geq q/(4\Delta)$ for some $t \geq 0$. If either $v \notin e_t$ or $\hat{P}(e_t) = 0$, then clearly $F_{t+1}(v) = F_t(v)$ and the statement is proven. Otherwise, by the definition of $\hat{P}$ it follows that

$F_{t+1}(v) = F_t(v) \cdot (1 - \hat{P}(e_t)) = F_t(v) \cdot (1 - P(e_t)) \geq q/(4\Delta)$, and again the statement is proven. The fact that $P(e_t) \leq 1/4$ is now a consequence of the previously proven fact and $q \geq 8\sqrt{\Delta}$, by (I.2):

$$P(e_t) = \frac{1}{\Delta + q} \cdot \frac{1}{F_t(u) \cdot F_t(v)} \leq \frac{1}{\Delta + q} \cdot \frac{1}{(q/(4\Delta))^2} \leq \frac{16\Delta}{q^2} \leq \frac{1}{4}.$$

□

To prove that $\hat{P}(e_t) = P(e_t)$ with high probability in Δ, we note that by their construction in Algorithm I.2, the values $\hat{P}(e_t)$ can only differ from $P(e_t)$ if the values of the variables $F_t(v)$ are too small, and in particular are close to their lower bound of $q/(4\Delta)$ guaranteed by Observation I.4.3.

Observation I.4.4. *If $e_t = (u, v)$ and $\min\{F_t(u),\ F_t(v)\} \geq q/(3\Delta)$, then $\hat{P}(e_t) = P(e_t)$.*

Proof. As $P(e_t) \leq 1/4$ by Observation I.4.3, we have $\min\{F_t(u), F_t(v)\} \cdot (1 - P(e_t)) \geq (q/(3\Delta)) \cdot (3/4) = q/(4\Delta)$, which, by the algorithm's definition, in turn implies that $\hat{P}(e_t) = P(e_t)$. □

Given Observation I.4.4 and Lemma I.4.2, it suffices to show that the probability that $F_t(v) < q/(3\Delta)$ is very small for all time steps t. As $F_{t+1}(v) \leq F_t(v)$, it is thus sufficient to bound this probability at the very last step, i.e., to bound the probability that $F(v) < q/(3\Delta)$, where $F(v) := F_m(v)$. In the following lemma, we identify a sufficient condition—Equation (I.3)— for the condition $F(v) \geq q/(3\Delta)$ (and thus also $\hat{P}(e) = P(e)$) to hold. In particular, we lower bound $F(v)$ by only focusing on the impact of neighbors of v on $P(e_{t_i})$ for edges $e_{t_i} \ni v$ and then applying the union bound.

Lemma I.4.5. *Let $e_{t_1} = (u_1, v), \ldots, e_{t_\ell} = (u_\ell, v)$ be the edges incident to v, arriving at times $t_1 < \cdots < t_\ell$. Let $S := \{u_i \in N(v) \mid u_i \notin M_{t_i}\}$ be those neighbors u_i that are not matched before time t_i when the edge $e_{t_i} = (u_i, v)$ arrives. Then,*

$$F(v) \geq 1 - \sum_{u_i \in S} \frac{1}{\Delta + q} \frac{1}{F_{t_i}(u_i)}.$$

As a consequence, $F(v) \geq q/(3\Delta)$ holds if

$$\sum_{u_i \in S} \frac{1}{\Delta + q} \frac{1}{F_{t_i}(u_i)} \leq \frac{\Delta}{\Delta + q/2}. \tag{I.3}$$

Proof. We look at how $F_t(v)$ develops throughout the run of Algorithm I.2. When edge $e_{t_i} = (u_i, v)$ arrives, the algorithm sets $F_{t_i+1}(v) \leftarrow F_{t_i}(v) \cdot (1 - \hat{P}(e_{t_i}))$, yielding

the following lower bound on $F_{t+1}(v)$, relying on $\hat{P}(e_{t_i}) \leq P(e_{t_i})$:

$$\begin{aligned} F_{t_i+1}(v) &\geq F_{t_i}(v) \cdot (1 - P(e_{t_i})) \\ &\geq F_{t_i}(v) \cdot \left(1 - \frac{1}{\Delta+q}\frac{1}{F_{t_i}(v)F_{t_i}(u)}\right) = F_{t_i}(v) - \frac{1}{\Delta+q}\frac{1}{F_{t_i}(u)}. \end{aligned}$$

Above, the second inequality is only an equality if both v, u_i are not matched before time t_i, and in particular $u_i \in S$. In the alternate case, we have that $\hat{P}(e_{t_i}) = P(e_{t_i}) = 0$, and so $F_{t+1}(v) = F_t(v)$. We conclude that $F_{t_i+1}(v) - F_{t_i}(v)$ can only be non-zero for times t_i with u_i previously unmatched ($u_i \in S$), in which case $F_{t_i+1}(v) - F_{t_i}(v) \geq -\frac{1}{\Delta+q}\frac{1}{F_{t_i}(u_i)}$. Since $F_1(v) = 1$ initially, the first part of the lemma follows by summing over all $u_i \in S$.

The second part of the lemma, whereby $F(v) \geq q/(3\Delta)$ provided Equation (I.3) holds, now follows from the first part and a simple calculation, using that $q \leq \Delta/4$ by Equation (I.2):

$$F(v) \geq 1 - \sum_{j=1}^{k} \frac{1}{\Delta+q}\frac{1}{F_{t_{i_j}}(u_{i_j})} \overset{\text{(I.3)}}{\geq} 1 - \frac{\Delta}{\Delta+q/2} = \frac{q/2}{\Delta+q/2} \geq q/(3\Delta)$$

which is what we wanted to prove. □

Similar arguments to those in Section I.4.1 can be used to prove that the sum $P := \sum_{u_i \in S} \frac{1}{\Delta+q}\frac{1}{F_{t_i}(u_i)}$ satisfies $\mathbb{E}[P] \leq \frac{\Delta}{\Delta+q}$. (This also follows from the subsequent martingale analysis later.) That the expectation of P is bounded away from the upper bound of $\frac{\Delta}{\Delta+q/2}$ required in (I.3) hints at using appropriate concentration inequalities, such as Chernoff-Hoeffding type bounds, to prove $P \leq \frac{\Delta}{\Delta+q/2}$ with high probability in Δ. Unfortunately, P is not a sum of independent or negatively correlated random variables in general, and therefore Chernoff-Hoeffding bounds are not applicable. Instead, in the subsequent sections we model the development of P as a martingale process, which will allow proving the desired concentration inequality without having to argue explicitly about correlations.

Our martingale process

Fix a vertex v. To prove that the sufficient condition $F(v) \geq q/(3\Delta)$ of inequality (I.3) holds often in general, we view the development of the left-hand-side of (I.3) as a martingale. For a time step t, define:

$$S_t := \{u_i \in N(v) \mid u_i \notin M_{\min\{t,t_i\}}\} \quad \text{and} \quad Y_{t-1} := \sum_{u_i \in S_t} \frac{1}{\Delta+q}\frac{1}{F_{\min\{t,t_i\}}(u_i)}.$$

Recall that t_i is the time step at which the edge $e_{t_i} = (v, u_i)$ arrives, and $N(v)$ is the final set of neighbors of v in the graph. Hence, S_t contains all neighbors of v in the final graph (including the future neighbors), except those neighbors that were

already matched by the time $\min\{t, t_i\}$. In particular, if $u_i \in S_{t_i}$, i.e., u_i was not matched by the time t_i it gets connected to v, it will remain inside all future sets S_t, for $t \geq t_i$. Also, notice that both S_t and Y_t are unknown to the algorithm at time t, as their definition requires "future" knowledge of the input graph, and that they are only used for the analysis.

With the above notation, $Y_0 = \frac{\deg(v)}{\Delta+q} \leq \frac{\Delta}{\Delta+q}$ and $Y := Y_m$ equals the left-hand-side of Equation (I.3). Y_{t-1} is determined by the independent random variables $X_1, \ldots, X_{t-1}$ sampled by Algorithm I.2. As we now show, $Y_0, \ldots, Y_m$ indeed form a martingale:

Lemma I.4.6. *$Y_0, \ldots, Y_m$ form a martingale w.r.t. the random variables $X_1, \ldots, X_m$. Furthermore, the difference $Y_t - Y_{t-1}$ is given by the following two cases:*

- *If e_t is added to M_{t+1}, which happens with probability $\hat{P}(e_t)$, then:*
$$Y_t - Y_{t-1} = -\frac{1}{\Delta+q} \sum_{u_i \in S_t \cap e_t} \frac{1}{F_t(u_i)}. \tag{I.4}$$
- *If instead e_t is not added to M_{t+1}, which happens with probability $1 - \hat{P}(e_t)$, then:*
$$Y_t - Y_{t-1} = \frac{1}{\Delta+q} \cdot \frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \sum_{u_i \in S_t \cap e_t} \frac{1}{F_t(u_i)}. \tag{I.5}$$

Proof. To prove the martingale property, we check the conditions given by Definition I.2.2. First, notice that fixing $X_1, \ldots, X_t$ in Algorithm I.2 determines the set S_t and the values of the random variables $F_{\min\{t,t_i\}}(u_i)$ used to define Y_t. Hence, Y_t is a function of $X_1, \ldots, X_t$. As $F_{\min\{t,t_i\}}(u_i) \geq q/(4\Delta)$, obviously $\mathbb{E}[Y_t] < \infty$.

It remains to show that $\mathbb{E}[Y_t \mid X_1, X_2, \ldots, X_{t-1}] = Y_{t-1}$. We first verify the claimed identities at (I.4) and (I.5). If the edge e_t arriving at time t is not incident to any $u_i \in S_t$ with $t < t_i$, then $Y_{t+1} = Y_t$ deterministically, and (I.4), (I.5) hold as their right hand sides are equal to 0. On the other hand, if e_t is incident to one or two such vertices $u_i \in S_t$, then we have two cases: e_t might be added to M_{t+1} or not. If e_t was matched, $S_{t+1} = S_t \setminus (S_t \cap e_t)$, so some terms are dropped from the sum and the identity (I.4) follows. Otherwise, if e_t was not matched, we have for any $u_i \in S_t \cap e_t$:

$$\begin{aligned}\frac{1}{\Delta+q}\left(\frac{1}{F_{t+1}(u_i)} - \frac{1}{F_t(u_i)}\right) &= \frac{1}{\Delta+q}\left(\frac{1}{F_t(u_i)(1-\hat{P}(e_t))} - \frac{1}{F_t(u_i)}\right) \\ &= \frac{1}{\Delta+q} \cdot \frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \cdot \frac{1}{F_t(u_i)},\end{aligned}$$

and identity (I.5) follows by summing the above for all $u_i \in S_t \cap e_t$.

Now $\mathbb{E}[Y_t \mid X_1, X_2, \ldots, X_{t-1}] = Y_{t-1}$ follows by direct computation using (I.4) and (I.5). Hence, all conditions of Definition I.2.2 are fulfilled, and indeed $Y_0, \ldots, Y_m$ forms a martingale w.r.t. $X_1, \ldots, X_m$. □

Bounding martingale parameters

We recall that our goal is to prove that Equation (I.3), i.e., that $Y = Y_m$ satisfies $Y \leq \frac{\Delta}{\Delta+q/2}$ with high probability (in Δ). As $Y_0 = \frac{\deg(v)}{\Delta+q} \leq \frac{\Delta}{\Delta+q}$ and trivially $Y \leq Y_0 + |Y - Y_0|$, it thus suffices to bound the difference between the first and last step of the martingale, $|Y - Y_0|$, as follows:

Fact I.4.7 (Sufficient Martingale Condition)**.** *Equation* (I.3) *holds if*

$$|Y - Y_0| \leq \frac{\Delta}{\Delta + q/2} - \frac{\Delta}{\Delta + q}. \tag{I.6}$$

Our idea for proving that inequality (I.6) holds often is to use specialized concentration inequalities for martingales, which do not require independence or explicit bounds on the positive correlation. Specifically, we will appeal to Freedman's inequality (see Lemma I.2.3). To this end, in the following two lemmas we upper bound this martingale's step size and observed variance.

Lemma I.4.8 (Step size)**.** *For all times t, we have $|Y_t - Y_{t-1}| \leq A$, where $A := 8/q$.*

Proof. By using the expressions for the difference $Y_t - Y_{t-1}$ from Lemma I.4.6, we obtain:

$$\begin{aligned}
|Y_t - Y_{t-1}| &\leq \frac{1}{\Delta+q} \cdot \max\left\{\frac{\hat{P}(e_t)}{1-\hat{P}(e_t)}, 1\right\} \cdot \sum_{u_i \in S_t \cap e_t} \frac{1}{F_t(u_i)} \\
&\leq \frac{1}{\Delta+q} \cdot \sum_{u_i \in S_t \cap e_t} \frac{1}{q/(4\Delta)} \\
&\leq \frac{1}{\Delta+q} \cdot \frac{2}{q/(4\Delta)} \\
&\leq 8/q.
\end{aligned}$$

For the second inequality, first notice that $\hat{P}(e_t) \leq P(e_t) \leq 1/2$ (by Observation I.4.3) which implies $\frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \leq 1$. Also, we have $F_t(u_i) \geq q/(4\Delta)$ (by Observation I.4.3) at any point of time in the algorithm. For the third inequality we used the trivial fact that $|S_t \cap e_t| \leq 2$. □

We next upper bound the observed variance W_m. While the following proof is computation-heavy, we note that all our following manipulations are straightforward, except for the insight that the hard lower bound $F(v) \geq q/(4\Delta)$ allows us to also bound $\sum_{e \in \delta(u)} \hat{P}(e)$.

Lemma I.4.9 (Observed Variance)**.** *For the martingale Y_t described above, we have:*

$$W_m := \sum_{t=1}^{m} \mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, \ldots, X_{t-1}] \leq \frac{128\Delta \ln \Delta}{q^2}. \tag{I.7}$$

Proof. Using the expression for $Y_t - Y_{t-1}$ (see Lemma I.4.6), we first have:

$$\mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, X_2, \ldots, X_{t-1}] =$$
$$\hat{P}(e_t) \cdot \left(\sum_{u_i \in S_t \cap e_t} \frac{1}{(\Delta + q) F_t(u_i)} \right)^2 +$$
$$(1 - \hat{P}(e_t)) \cdot \left(\sum_{u_i \in S_t \cap e_t} \frac{\hat{P}(e_t)}{(\Delta + q) F_t(u_i)(1 - \hat{P}(e_t))} \right)^2 .$$

Note that the $\hat{P}(e_t)$'s and $F_t(u_i)$'s depend on the variables $X_1, X_2, \ldots, X_{t-1}$ we are conditioning on, and that we will show the bound on the observed variance in *any* execution of the algorithm. The above sums contain either one or two terms, as $|S_t \cap e_t| \leq 2$. By using the elementary inequality $(a+b)^2 \leq 2a^2 + 2b^2$, we obtain the following upper bound:

$$\mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, X_2, \ldots, X_{t-1}] \leq$$
$$2\hat{P}(e_t) \cdot \sum_{u_i \in S_t \cap e_t} \left(\frac{1}{(\Delta + q) F_t(u_i)} \right)^2 +$$
$$2(1 - \hat{P}(e_t)) \cdot \sum_{u_i \in S_t \cap e_t} \left(\frac{\hat{P}(e_t)}{(\Delta + q) F_t(u_i)(1 - \hat{P}(e_t))} \right)^2 .$$

The above expression can be rewritten compactly by factoring out $\frac{2\hat{P}(e_t)}{(\Delta+q)^2 \cdot (F_t(u_i))^2}$, which gives:

$$\mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, X_2, \ldots, X_{t-1}] \leq \sum_{u_i \in S_t \cap e_t} \frac{2\hat{P}(e_t)}{(\Delta + q)^2 (F_t(u_i))^2} \left(1 + \frac{\hat{P}(e_t)}{1 - \hat{P}(e_t)} \right).$$

Using Observation I.4.3, we have $\hat{P}(e_t) \leq P(e_t) \leq 1/2$ and thus $1 + \frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \leq 2$, but also $F_t(u_i) \geq q/(4\Delta)$ for any $u_i \in S_t \cap e_t$. Additionally we note that $|S_t \cap e_t| \leq 2$, and so we have:

$$\mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, X_2, \ldots, X_{t-1}] \leq \sum_{u_i \in S_t \cap e_t} \frac{2\hat{P}(e_t)}{(\Delta + q)^2} \cdot \frac{16\Delta^2}{q^2} \cdot 2 \leq \frac{128\hat{P}(e_t)}{q^2}.$$

By summing this inequality over all t we will obtain an upper bound for W_m. Notice that an edge e_t is thereby summed over on the right hand side only if it is incident to some vertex $u_i \in S_t$ at some time step t (see Figure I.3). As $S_t \subseteq S_0 = N(v)$, we obtain the following upper bound, double counting edges e_t that connect two distinct vertices of S_t (for example edge (u_2, u_3) in Figure I.3):

$$W_m := \sum_{t=1}^{m} \mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, X_2, \ldots, X_{t-1}] \leq \sum_{u_i \in N(v)} \sum_{e_t \in \delta(u_i)} \frac{128\hat{P}(e_t)}{q^2}. \quad \text{(I.8)}$$

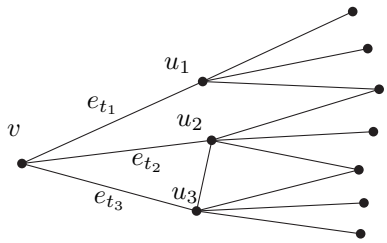


Figure I.3: The configuration described in Lemma I.4.5, i.e., the 2-hop neighborhood of v.
The (at most) Δ^2 edges drawn correspond to the only non-trivial steps of the martingale.

To upper bound the inner sum, fix a vertex $u_i \in N(v)$ and note that:

$$\frac{q}{4\Delta} \leq F_m(u_i) = \prod_{e \in \delta(u_i)} (1 - \hat{P}(e)) \leq \exp\left(- \sum_{e \in \delta(u_i)} \hat{P}(e) \right),$$

which implies that $\sum_{e \in \delta(u_i)} \hat{P}(e) \leq \ln\left(\frac{4\Delta}{q}\right) \leq \ln(\Delta)$ (using that $q \geq 8\sqrt{\Delta} \geq 8 \geq 4$). By plugging the above into (I.8) and using the fact that $|N(v)| \leq \Delta$, we finally can conclude the proof of Lemma I.4.9:

$$W_m := \sum_{t=1}^{m} \mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_1, X_2, \ldots, X_{t-1}] \leq \frac{128\Delta \ln \Delta}{q^2}.$$

□

Conclusion of the analysis

Having upper bounded both step size and variance of the martingale $Y_0, Y_1, \ldots, Y_m =: Y$, we are now ready to leverage Freedman's inequality to prove that with high probability (in Δ), our desired upper bound on $|Y - Y_0|$ of inequality (I.6) holds, and hence $F(v) \geq q/(3\Delta)$.

Lemma I.4.10. $\Pr[F(v) < q/(3\Delta)] \leq 2\Delta^{-3}$.

Proof. Let $\lambda := q/(3\Delta)$. By Fact I.4.7 and Lemma I.4.5, we have that

$$\Pr[F(v) < q/(3\Delta)] \leq \Pr\left[|Y - Y_0| \geq \frac{\Delta}{\Delta + q/2} - \frac{\Delta}{\Delta + q}\right] \leq \Pr[|Y - Y_0| \geq \lambda]. \quad \text{(I.9)}$$

Above, the second inequality used that

$$\frac{q}{3\Delta} \leq \frac{\Delta}{\Delta + q/2} - \frac{\Delta}{\Delta + q},$$

which, by taking a common denominator, is seen to be equivalent to $3\Delta q + q^2 \leq \Delta^2$, which follows directly from $q \leq \Delta/4$ (see (I.2)).

Next, to upper bound the RHS of Equation (I.9), we appeal to Freedman's inequality (Lemma I.2.3). Letting $\sigma^2 := \frac{128\Delta \ln \Delta}{q^2}$ and $A := 8/q$ (as in Lemmas I.4.8 and I.4.9) and λ as above, we have that:

$$\begin{aligned}
\Pr[|Y - Y_0| \geq \lambda] &\leq 2\exp\left(-\frac{\lambda^2}{2(\sigma^2 + A\lambda)}\right) \\
&= 2\exp\left(-\frac{(\frac{q}{3\Delta})^2}{2\left(\frac{128\Delta \ln \Delta}{q^2} + \frac{8}{q} \cdot \frac{q}{3\Delta}\right)}\right) \\
&= 2\exp\left(-\frac{200 \ln \Delta}{18\left(128/200 + 8/3 \cdot \Delta^{-1/2}\right)}\right) \\
&\leq 2\exp(-3\ln\Delta) \\
&= 2\Delta^{-3},
\end{aligned}$$

where for the second-to-last equality we used the definition of $q = \sqrt{200} \cdot \Delta^{3/4} \ln^{1/2} \Delta$. Combining the above with Equation (I.9), the lemma follows. □

To obtain the statement of Theorem I.1.3, it remains to put the pieces together. With high probability in Δ, the inequality from Lemma I.4.5 holds for both vertices incident at an edge e. Also, by a sequence of other lemmas we know that this inequality implies that $\hat{P}(e) = P(e)$ (with high probability in Δ). Intuitively, this should guarantee the right marginal $1/(\Delta + O(q))$ of matching e. More formally, we now complete the proof of our main technical contribution:

Theorem I.1.3. *There exists an online matching algorithm that on graphs with known maximum degree Δ, outputs a random matching M satisfying*

$$\Pr[e \in M] \geq \frac{1}{\Delta + \Theta(\Delta^{3/4} \log^{1/2} \Delta)} = \frac{1}{(1 + o(1)) \cdot \Delta} \qquad \forall e \in E.$$

Proof. We prove that Algorithm I.2 with q as in this section is such an algorithm. Fix an edge $e_t = (u, v)$. The marginal probability that e_t is matched by Algorithm I.2 is given by $\Pr[X_t < \hat{P}(e_t)]$. We note that for the event $X_t < \hat{P}(e_t)$, we need that both $X_t < P(e_t)$ and $\hat{P}(e_t) = P(e_t)$ (else $\hat{P}(e_t) = 0$ and so trivially $X_t \geq \hat{P}(e_t)$).

We therefore have that

$$\begin{aligned}
\Pr[X_t < \hat{P}(e_t)] &= \Pr\left[(X_t < P(e_t)) \wedge \left(\hat{P}(e_t) = P(e_t)\right)\right] \\
&= \Pr[X_t < P(e_t)] - \Pr\left[(X_t < P(e_t)) \wedge \left(\hat{P}(e_t) \neq P(e_t)\right)\right] \\
&\geq \Pr[X_t < P(e_t)] - \Pr[\hat{P}(e_t) \neq P(e_t)] \\
&\geq \Pr[X_t < P(e_t)] - (\Pr[F(v) < q/(3\Delta)] + \Pr[F(u) < q/(3\Delta)]) \\
&\geq \frac{1}{\Delta + q} - 4\Delta^{-3} \\
&\geq \frac{1}{\Delta + 4q}.
\end{aligned}$$

Above, the second and third inequalities follow from Lemma I.4.2 and Lemma I.4.10 together with union bound over $u, v \in e_t$. The last inequality, on the other hand, which is equivalent to $\frac{3q}{(\Delta+q)(\Delta+4q)} \geq 4\Delta^{-3}$, which clearly holds for large enough Δ, and can be verified to hold for all $\Delta \geq 1$, using $8\sqrt{\Delta} \leq q \leq \Delta/4$, by (I.2). □

From Matching to Edge Coloring We are finally ready to prove our main result, the existence of an online $\Delta(1 + o(1))$-edge-coloring algorithm. To do so, it suffices to combine our online matching algorithm of Theorem I.1.3 with the known reduction from online edge coloring to online matching given by Lemma I.2.1, or alternatively, our strengthening of the latter in Lemma I.5.15. Combining these, we obtain the following quantitative result.

Theorem I.4.11. *There exists an online edge-coloring algorithm which, on n-vertex graphs of known maximum degree $\Delta = \omega(\log n)$, outputs with high probability (in n) a valid edge coloring using $\Delta + q'$ colors, where:*

$$q' = O(\Delta^{3/4} \log^{1/2} \Delta + \Delta^{2/3} \log^{1/3} n).$$

Proof. By Theorem I.1.3, there exists an online matching algorithm matching each edge with probability at least $1/(\alpha\Delta)$, for $\alpha := 1 + O(\Delta^{-1/4} \log^{1/2} \Delta)$. But by Lemma I.5.15, such an online matching algorithm can be used to obtain an $(\alpha + O(\Delta^{-1/3} \log^{1/3} n) \cdot \Delta$-coloring algorithm (assuming $\Delta = \omega(\log n)$). Combining Theorem I.1.3 and Lemma I.5.15, we therefore obtain an online edge coloring algorithm that, w.h.p. in n, on graphs of degree $\Delta = \omega(\log n)$, uses the claimed number of colors:

$$(\alpha + O(\Delta^{-1/3} \log^{1/3} n)) \cdot \Delta = \Delta + O(\Delta^{3/4} \log^{1/2} \Delta + \Delta^{2/3} \log^{1/3} n).$$

□

APPENDIX

I.5 Online List Edge Coloring and Local Edge Coloring

In this section we present a number of applications that follow from our Algorithm I.2. In particular, we show generalizations to *online list edge coloring* (Theorem I.5.1, in Section I.5.3) and *online local edge coloring* (Theorem I.5.2, in Section I.5.4).

In the online *list* edge coloring problem, every online arriving edge e presents a list $L(e) \subseteq \mathbb{N}$ of colors that can be used for coloring e. Unlike in the classical setting, this list $L(e)$ is not necessarily of the form $\{1, \ldots, \Delta + q\}$ for some $q \geq 1$, but can instead be arbitrary. The objective of an algorithm is to provide a valid coloring of all edges with each edge e assigned color in its list, $c(e) \in L(e)$, and no vertex having two incident edges assigned the same color. Our result for this problem is given by:

Theorem I.5.1 (Online List Edge Coloring)**.** *There exists an online list edge-coloring algorithm which, on n-vertex graphs of known maximum degree Δ, outputs with high probability (in n) a valid list-edge coloring, provided all lists $L(e)$ satisfy $|L(e)| \geq \Delta + q$, for*

$$q := 10^{24} \ln n + 10^4 \cdot \left(\Delta^{3/4} \ln^{1/2} \Delta + \Delta^{2/3} \ln^{1/3} n \right).$$

In particular, if $\Delta = \omega(\log n)$, then lists of size $\Delta + q$ with $q = o(\Delta)$ suffice.

In the online *local* edge coloring problem, the setting is the same as in the classical version. However, we now aim to color each edge $e = (u, v)$ using a color of index $c(e)$ that is not much larger than the max degree of its endpoints, $d_{\max}(e) := \max\{\deg(u), \deg(v)\}$. Here we prove the following result:

Theorem I.5.2 (Online Local Edge Coloring)**.** *There exists an online edge-coloring algorithm which, on n-vertex graphs with* a priori *known degree sequence* $\{\deg(v) \mid v \in V\}$,[4] *computes, with high probability (in n), an edge coloring $c : E \to \mathbb{N}$ that colors each edge e using a color $c(e)$ which satisfies:*

$$c(e) \leq d_{max}(e) + 2 \cdot 10^{24} \ln n + 10^5 \cdot \left(d_{max}^{3/4}(e) \ln^{1/2} d_{max}(e) + d_{max}^{2/3}(e) \ln^{1/3} n \right).$$

In particular, if $d_{max}(e) = \omega(\log n)$, then $c(e) \leq d_{\max}(e) \cdot (1 + o(1))$.

To obtain both generalizations of Theorem I.1.2 above, we rely on an online coloring subroutine, Algorithm I.3, which we provide in the following section. This algorithm colors a (potentially strict) subset of the edges of the graph, and assigns each colored edge e a color from some individual (small) list of colors $\ell(e)$, revealed when e arrives. Under the mild technical condition (which holds with high

[4] This theorem also holds if $\deg(v)$ are only *upper bounds* for the true degrees, in which case the guarantees of the theorem will be with respect to these upper bounds.

probability for our invocations of this algorithm in later sections) that $|\ell(e)| \approx \lambda$ for some appropriate parameter λ, the subroutine reduces the maximum degree Δ of the remaining uncolored subgraph by $\approx \lambda$. In other words, for all high-degree vertices in U (those with degree $\approx \Delta$), the algorithm colors $\approx \lambda$ of their incident edges. This subroutine is heavily inspired by [CPW19, Algorithm 4], which uses (instead of our Algorithm I.2) an older online matching rounding algorithm, called MARKING [CW18; Waj20], for sampling matchings. Unlike our subroutine which works under edge arrivals, the previous [CPW19, Algorithm 4] is restricted to one-sided vertex arrivals, because MARKING is also restricted to this regime. Moreover, our algorithm's more general ability to color from lists $\ell(e)$ for each edge e allows us to obtain both the list edge coloring result and local edge coloring results, which could not be obtained from [CPW19, Algorithm 4].

In Section I.5.2, we present Line 9, which applies Algorithm I.3 successively on monotonically decreasing subgraphs of G to color edges. This coloring algorithm is very flexible, in the sense that every arriving edge e presents a personalized list $L(e)$ of colors that can be used to color it. This algorithm forms the underpinning of both theorems above, proven in later sections.

I.5.1 Subroutine to Reduce the Maximum Degree

In this section we provide an algorithmic subroutine for reducing the uncolored maximum degree. This will be used for our list and local edge coloring results, as well as our improvement on the reduction from edge coloring to matching of Lemma I.2.1.

The psueocode of this algorithmic suroutine is given in Algorithm I.3, and it works as follows. Its input is a graph U with n vertices and maximum degree upper bounded by some $\Delta(U)$ arriving online edge-by-edge. (We think of U as the *uncolored* subgraph of an initial graph G on the same vertex set.) Let $\lambda := \Delta(U)^{2/3} \ln^{1/3} n$. We call a vertex $v \in V$ *dense* if $\deg_U(v) \geq \Delta(U) - \lambda$. For any arriving edge e, the algorithm receives a list $\ell(e)$ of available colors. If e is incident to a dense vertex v, the list $\ell(e)$ is guaranteed to have size $|\ell(e)| \approx \lambda$. For each new color revealed, we run a copy of Algorithm I.2, guaranteeing that each edge is matched (and hence) colored by each of the colors $c \in \ell(e)$ with probability $\approx \frac{1}{\Delta}$.

As we now show, if the maximum degree of U is sufficiently large(r than $\ln n$), the above algorithm guarantees with high probability that dense vertices have $\approx \lambda$ edges colored by this algorithm, and therefore the maximum degree of the remaining uncolored graph decreases by roughly λ as well.

Theorem I.5.3. *Every edge e colored by Algorithm I.3 is assigned a color $c \in \ell(e)$. Moreover, after running Algorithm I.3 on U with $\Delta(U) \geq 10^{24} \ln n$, and using Algorithm I.2 with $q := 10^2 \cdot \Delta(U)^{3/4} \ln^{1/2} \Delta(U)$, with the necessary promises, the maximum degree of the remaining uncolored subgraph $U' \subseteq U$ is upper bounded by $\Delta(U') := \Delta(U) - \lambda + 2\lambda \cdot \frac{q+\lambda}{\Delta+q} + 6\sqrt{\lambda \ln n} \leq \Delta(U) - 0.9\lambda$ with probability at least $1 - \frac{1}{n^{15}}$.*

Algorithm I.3: Coloring Algorithm Reducing Maximum Degree

Input: Graph U with n vertices, arriving edge-by-edge. Each edge e arrives with list of colors $\ell(e)$.
Promise: A value $\Delta(U)$ upper bounding the maximum degree of U is given.
Promise: For any edge e incident to a dense vertex, we have

$$\lambda \leq |\ell(e)| \leq \lambda + 10\sqrt{\lambda \ln n}, \text{ for } \lambda := \Delta(U)^{2/3} \ln^{1/3} n.$$

Output: Coloring of a subset of edges of U.

When edge e together with list $\ell(e)$ arrives:
for $c \in \ell(e)$ **do**
    If c was never seen before, launch a new instance of Algorithm I.2 corresponding to c on U, using $\Delta(U)$ as an upper bound for the maximum degree of U.
    Input the edge e to the instance of Algorithm I.2 corresponding to c.
    If e was matched by Algorithm I.2 and was not already colored, color e with color c.

Before proving the above theorem, it will be convenient to present some useful inequalities in a separate lemma, whose technical proof (which are easy to verify hold asymptotically, but also hold for all choices of $n \geq 2$ due to the large constants chosen) is deferred to the end of the section:

Fact I.5.4. *Suppose $\Delta(U) \geq 10^{24} \ln n$ and define the following parameters:*

$$\begin{aligned} d &:= \Delta(U), \\ q &:= 10^2 \cdot d^{3/4} \ln^{1/2} d \\ \lambda &:= d^{2/3} \ln^{1/3} n \\ \mu &:= (d-\lambda) \cdot \frac{\lambda}{d+q} \left(1 - \frac{\lambda}{d+q}\right). \end{aligned}$$

Then, these parameters satisfy the following inequalities:

$$\mu \geq \lambda - 2\lambda \cdot \frac{q+\lambda}{d+q} \tag{I.10}$$

$$0.1\lambda \geq 2\lambda \cdot \frac{q+\lambda}{d+q} + 6\sqrt{\lambda \ln n} \tag{I.11}$$

$$\mu \geq 30 \ln n. \tag{I.12}$$

Proof of Theorem I.5.3. That each edge e is assigned a color (if any) from $\ell(e)$ is immediate from the algorithm's description. The "meat" of the proof is therefore in

proving that removal of the edges colored yields a subgraph of maximum degree as low as claimed in the theorem statement.

Consider a dense vertex v of U, i.e., with degree $\deg_U(v) \geq d - \lambda$. (If no such vertex exists, the statement to prove follows trivially.) By the proof of Theorem I.1.3, each edge e incident to v is picked by color $c \in \ell(e)$ with probability at least $\frac{1}{d+4q'} \geq \frac{1}{d+q}$, for $q' = \sqrt{200}d^{3/4}\ln^{1/2} d$, where $4q' \leq q$. Let X_e be the indicator variable for the event that e is colored by Algorithm I.3 with any color $c \in \ell(e)$. As $|\ell(e)| \geq \lambda$ for dense vertices, we have by the first two terms of the Taylor expansion of $\exp(-\lambda/(d+q))$ for $\lambda/(d+q) \leq 1$ (as in our case):

$$\Pr[X_e = 1] \geq 1 - \left(1 - \frac{1}{d+q}\right)^{\lambda} \geq 1 - \exp\left(-\frac{\lambda}{d+q}\right) \geq \frac{\lambda}{d+q} - \left(\frac{\lambda}{d+q}\right)^2.$$

Let $X := \sum_{e\in\delta(v)} X_e$ be the number of edges incident to v that are colored by Algorithm I.3. We wish to argue that X is not much less than the following lower bound $E[X] \geq \mu := (d-\lambda)\cdot\frac{\lambda}{d+q}\left(1 - \frac{\lambda}{d+q}\right)$ on its expectation. (This lower bound follows by our lower bound on $\Pr[X_e = 1]$, linearity of expectation, and the lower bound on the number of edges of dense vertices.)

We now argue that with high probability in n, the expectation $\mathbb{E}[X]$ does not fall short of this lower bound of μ. This would follow easily by standard Chernoff bounds if the X_e were independent, though they clearly are not. However, we can interpret these variables as indicator variables of a balls and bins process, which are *negatively associated (NA)*, and hence admit the same Chernoff bounds as independent random variables [DR96; Waj17]. In more detail, we have a ball for each color c, and each ball (color) c falls into a (single) bin corresponding to either an edge e of v or a dummy bin, depending on whether or not e is matched by the c^{th} copy of Algorithm I.2, which happens independently (but not i.i.d) for different c. The values X_e are therefore indicators for whether bin e is non-empty, and these random variables are NA [Waj20, Corollary 2.4.6]. Consequently, the sum X of the NA (but not independent) random variables X_e, which satisfies $\mathbb{E}[X] \geq \mu$, also satisfies the following standard Chernoff bound for any $\varepsilon < 1$:

$$\Pr\left[X \leq (1-\epsilon)\cdot\mu\right] \leq \exp\left(-\frac{\varepsilon^2\cdot\mu}{2}\right). \tag{I.13}$$

Fix $\varepsilon := \sqrt{\frac{30\ln n}{\mu}}$. By Equation (I.12) from Fact I.5.4, have that $\varepsilon \in [0,1]$. On the other hand, as we shall soon see, we also have that

$$(1-\varepsilon)\cdot\mu \geq \lambda - 2\lambda\cdot\frac{q+\lambda}{\Delta+q} - 6\sqrt{\lambda\ln n}, \tag{I.14}$$

To see this, first note that, as $\mu = \lambda\cdot\frac{d-\lambda}{d+q}\cdot\left(1-\frac{\lambda}{d+q}\right) \leq \lambda$, we have that $\varepsilon\cdot\mu \leq \sqrt{30\lambda\ln n} \leq 6\sqrt{\lambda\ln n}$. Furthermore, $\mu \geq \lambda - 2\lambda\cdot\frac{q+\lambda}{\Delta+q}$, by (I.10) from Fact I.5.4. We

obtain (I.14) by summing up these last two inequalities. Consequently, combining (I.13) and (I.14) and using our choice of ε, we have that

$$\Pr\left[X \leq \lambda - 2\lambda \cdot \frac{q+\lambda}{\Delta+q} - 6\sqrt{\lambda \ln n}\right] \leq \Pr[X \leq (1-\varepsilon)\cdot\mu] \leq \frac{1}{n^{15}}.$$

The claimed upper bound on the maximum degree of U' then follows, since this upper bound holds for all non-dense vertices in U (which is a super graph of U'), together with union bound over all dense vertices v, which all have degree at most $\Delta(U)$ in U, and so, with high probability in n, in the new graph U' all vertices have degree at most $\Delta(U) - \left(\lambda - 2\lambda \cdot \frac{q+\lambda}{\Delta+q} - 6\sqrt{\lambda \ln n}\right) \leq \Delta(U) - 0.9\lambda$, where the last inequality follows from (I.11) from Fact I.5.4. □

The inequalities from Fact I.5.4 are obtained by direct computation as follows:

Proof of Fact I.5.4. (I.10) follows easily:

$$\begin{aligned}\mu &= (d-\lambda)\cdot\frac{\lambda}{d+q}\left(1-\frac{\lambda}{d+q}\right) = \lambda\cdot\left(1-\frac{q+\lambda}{d+q}\right)\left(1-\frac{\lambda}{d+q}\right)\\ &\geq \lambda - \lambda\cdot\frac{q+\lambda}{d+q} - \lambda\cdot\frac{\lambda}{d+q} \geq \lambda - 2\lambda\cdot\frac{q+\lambda}{d+q}.\end{aligned}$$

To get (I.11) it suffices to show:

$$2\lambda\cdot\frac{q+\lambda}{d+q} \leq 0.05\lambda \tag{I.15}$$

$$6\sqrt{\lambda\ln n} \leq 0.05\lambda. \tag{I.16}$$

For (I.15), notice that:

$$2\cdot\frac{q+\lambda}{d+q} \leq 2\cdot\frac{q+\lambda}{d} = \frac{2\cdot 10^2\cdot d^{3/4}\ln^{1/2} d}{d} + \frac{2\cdot d^{2/3}\ln^{1/3} n}{d}.$$

The first term can be upper bounded by 0.025 for any $n \geq 2$ given $d \geq 10^{24}\ln n$. The second term can be upper bounded by $\frac{2\ln^{1/3} n}{d^{1/3}} \leq 2/10^8 \leq 0.025$, since $d \geq 10^{24}\ln n$. Summing the upper bounds for the two terms gives (I.15). Inequality (I.16) follows from $\sqrt{\lambda/\ln n} \geq 10^8 \geq 120$, again since $d \geq 10^{24}\ln n$ and so $\lambda = d^{2/3}\ln^{1/3} n \geq 10^{16}\ln n$.

It remains to prove (I.12). By (I.10) and (I.11), we have that $\mu \geq 0.9\lambda$. And indeed, again using that $d \geq 10^{24}\ln n$ and so $\lambda \geq 10^{16}\ln n$, we obtain the desired inequality, as $\mu \geq 0.9\cdot 10^{16}\ln n \geq 30\ln n$. □

We now turn to using this subroutine to completely color a given graph revealed online.

I.5.2 Generic Coloring Algorithm

Strategy For subsequent applications, we consider graphs G with a maximum degree known to be at most Δ, with edges arriving online one by one. Moreover, every arriving edge e provides a list of available colors $L(e)$. We will repeatedly apply Algorithm I.3, consuming different sublists $\ell(e) \subseteq L(e)$ of colors per phase, to reduce the maximum degree of the currently uncolored graph $U \subseteq G$, by coloring subsets of the edges of U. The following definition introduces the relevant parameters and is used to define Line 9:

Definition I.5.5 (Degree Sequence)**.** Let $d_0 := \Delta$ be (an upper bound on) the maximum degree of a graph G with n vertices. For $i \geq 0$, we define the following:

$$\lambda_i := d_i^{2/3} \ln^{1/3} n,$$
$$q_i := 10^2 \cdot d_i^{3/4} \ln^{1/2} d_i,$$
$$d_{i+1} := d_i - \lambda_i + 2\lambda_i \cdot \frac{q_i + \lambda_i}{d_i + q_i} + 6\sqrt{\lambda_i \ln n} \leq d_i.$$

Let f be the minimal value for which $d_f < 10^{24} \ln n$. (Such an f exists, since $d_i \geq 10^{24} \ln n$, then by Fact I.5.4, $d_{i+1} \leq d_i - 0.9\lambda_i \leq d_i - 1$.) We call the parameters $D(d_0) := \{d_i : 0 \leq i \leq f + 1\}$ the *degree sequence of* d_0.

Algorithm I.4: Generic Coloring Algorithm

1 **Input:** Graph G with n vertices, arriving edge-by-edge together with lists $L(e)$ of available colors. We use the notations introduced in Definition I.5.5, and denote by $\mathcal{C} := \cup_e L(e)$ the set of all colors.
2 **Output**: Coloring of a subset of edges of G.

3 Let $C_0, \ldots, C_{f+1}$ be a partitioning of $\mathcal{C}$ which is computed online.
4 Set $U_0 \leftarrow G$.
5 **for** *phase* $i \in \{0, \ldots, f\}$ **do**
6 Apply Algorithm I.3 (online) on the currently uncolored graph U_i.
7 For any edge e incident to a *dense* vertex in U_i (i.e., having degree $\geq d_i - \lambda_i$), use the sublist $\ell^i(e) := L(e) \cap C_i$ to input online to Algorithm I.3.
8 Set $U_{i+1} \leftarrow U_i \setminus \{\text{edges colored by Algorithm I.3 in phase i}\}$
9 Try to color the final uncolored graph U_{f+1} using Greedy with the remaining lists of available colors $\ell^{f+1}(e) := L(e) \cap C_{f+1}$ for each edge e.

Let $\mathcal{C} := \cup_{e \in G} L(e)$ be the set of all colors. Note that this set is *a priori* unknown to the online algorithm and is only revealed indirectly through the lists $L(e)$ of the arriving edges. Line 9 partitions this set into $f + 2$ subsets of colors $C_0, \ldots, C_{f+1}$

online. More concretely, whenever the arriving list $L(e)$ of an edge e contains a color $c \in L(e)$ which is seen for the first time, it decides online to which of the sets $C_0, \dots, C_{f+1}$ color c will be assigned. *We insist on the fact that this choice can be made arbitrarily depending on the application. However, there is a restriction which we discuss in the next paragraph.*

The *i-th phase* is the iteration of Line 9 on the uncolored subgraph U_i. The partitioning of $\mathcal{C}$ into $C_0, \dots, C_{f+1}$ defines the colors to be used in each phase. Thereby, $\ell^i(e) := L(e) \cap C^i$ is the chosen sublist of $L(e)$ which is inputted online to Algorithm I.3 for edge e during the execution of the i-th phase. Let $L^i(e) := L(e) \setminus \cup_{j=0}^{i-1} \ell^j(e)$ be the set of colors which are still available to use for e during the i-th phase. For all edges e, the sublists $\ell^i(e) \subseteq L^i(e)$ must have the size required to apply Theorem I.5.3, that is, $\lambda_i \leq |\ell^i(e)| \leq \lambda_i + 10\sqrt{\lambda_i \ln n}$. This motivates the following:

Definition I.5.6 (Admissible Partitioning of Colors)**.** Given a fixed input of Line 9, let $C_0, \dots, C_{f+1}$ be a partitioning of the set of colors $\mathcal{C} := \cup_{e \in G} L(e)$ which is computed online. We say that the partitioning $C_0, \dots, C_{f+1}$ is *admissible* if for any edge e and phase $i \in \{0, \dots, f\}$ in which e is incident to a *dense* vertex in U_i (i.e., having degree $\geq d_i - \lambda_i$), we have that $\ell^i(e) := L(e) \cap C_i$ satisfies:

$$\lambda_i \leq |\ell^i(e)| \leq \lambda_i + 10\sqrt{\lambda_i \ln n}.$$

By successively applying Algorithm I.3 (starting with the initial graph $U_0 := G$), as done in Line 9, and *using an admissible (online) partitioning of $\mathcal{C}$ into $C_0, \dots, C_{f+1}$ as defined in Definition I.5.6*, one obtains a sequence of subgraphs $U_0 \supseteq U_1 \supseteq \cdots \supseteq U_{f+1}$ of G, such that by successive applications of Theorem I.5.3 we obtain the following:

Lemma I.5.7. *With high probability in n, the graphs U_i computed online by Line 9 (consisting of yet uncolored edges) have their maximum degree bounded by d_i. Moreover, for $i \leq f$ we have $d_i \geq 10^{24} \ln n$.*

Proof. Let good_i be the event that U_i has its degree upper bounded by d_i, where $i \in \{0, \dots, f\}$. Clearly, good_0 holds by the fact that d_0 is an upper bound on the maximum graph of the initial graph G. Assuming good_i holds, consider the i-th phase of Line 9. Then, by Theorem I.5.3, the probability that good_{i+1} holds is at least $1 - \frac{1}{n^{10}}$. Hence:

$$\Pr[\text{good}_{i+1} \mid \text{good}_i] \geq 1 - \frac{1}{n^{10}}. \tag{I.17}$$

By induction it easily follows that $\Pr[\text{good}_i] \geq 1 - \frac{i}{n^{10}}$ for any $i \in \{0, \dots, f\}$. For $i = 0$ this is true with with probability 1, and for the induction step we use (I.17) together with the induction hypothesis to obtain:

$$\Pr[\text{good}_{i+1}] \geq \Pr[\text{good}_{i+1} \mid \text{good}_i] \cdot \Pr[\text{good}_i] \geq \left(1 - \frac{1}{n^{10}}\right) \cdot \left(1 - \frac{i}{n^{10}}\right) \geq 1 - \frac{i+1}{n^{10}}.$$

In particular, for the complementary events we have $\Pr[\overline{\text{good}_i}] \leq \frac{1}{n^9}$ for any i, and the lemma follows by union bound over the at most n possible values of i:

$$\Pr\left[\cup_{i\in\{0,\dots f\}} \overline{\text{good}_i}\right] \leq n \cdot \frac{1}{n^9} \leq \frac{1}{n^8}.$$

□

Analysis We have designed a generic coloring algorithm which, with high probability, successively reduces the maximum degree of the uncolored subgraphs U_i until their maximum degree drops below $O(\log n)$. It remains to argue how to ensure the following properties required implicitly by Line 9:

- The lists $L^i(e)$ of remaining colors need to have sufficiently large size to allow extracting the sublists $\ell^i(e) \subseteq L^i(e)$, such that $|\ell^i(e)| \geq \lambda_i$ as required by the application of Algorithm I.3 inside Line 9.
- In particular, to color U_{f+1} successfully using Greedy, one needs to ensure that $L^{f+1}(e) \geq 2d_{f+1}$.

To obtain these guarantees, we maintain by induction, throughout all phases i, the property $|L^i(e)| \geq d_i + a_i$ for edges connected to dense vertices, where a_i is a large enough slack. We define these slacks precisely:

Definition I.5.8 (Slack Sequence)**.** Let $d_0 \geq 1$ and introduce the degree sequence $D(d_0)$ of d_0, and the parameters f, λ_i, q_i as in Definition I.5.5. We define the *slack sequence* $Sl(d_0) := \{a_i : 0 \leq i \leq f+1\}$ where:

$$a_{f+1} := 10^{24} \ln n > d_{f+1}$$
$$a_i := a_{i+1} + 2\lambda_i \cdot \frac{q_i + \lambda_i}{d_i + q_i} + 16\sqrt{\lambda_i \ln n}.$$

As anticipated, we prove by induction that these slacks fulfill the required guarantees:

Lemma I.5.9. *Assume that $|L(e)| \geq 2 \cdot 10^{24} \ln n$ for all edges e. Furthermore, assume that, before the execution of the i-th phase of Line 9, for any edge e connected to a dense vertex, $|L^i(e)| \geq d_i + a_i$. Then, after the execution of the i-th phase, one has $|L^{i+1}(e)| = |L^i(e) \setminus \ell^i(e)| \geq d_{i+1} + a_{i+1}$. Furthermore, executing Greedy on U_{f+1} in Line 9 is possible, as $|L^{f+1}(e)| \geq 2d_{f+1}$ for all edges $e \in U_{f+1}$.*

Proof. Using $|\ell^i(e)| \leq \lambda_i + 10\sqrt{\lambda_i \ln n}$, we obtain:

$$\begin{aligned}|L^{i+1}(e)| = |L^i(e)| - |\ell^i(e)| &\geq d_i + a_i - |\ell^i(e)| \\ &\geq d_i + a_i - (\lambda_i + 10\sqrt{\lambda_i \ln n}) = d_{i+1} + a_{i+1},\end{aligned}$$

where the last inequality follows by the definitions of a_i and d_{i+1} (see Definition I.5.5 and Definition I.5.8). As for the property $|L^{f+1}(e)| \geq 2d_{f+1}$, if e was ever connected to a dense vertex in some phase i, then the property follows because $|L^{f+1}(e)| \geq d_{f+1} + a_{f+1} \geq 2d_{f+1}$, where the last inequality is due to $a_{f+1} > d_{f+1}$ (see Definition I.5.8). If on the contrary e was never connected to a dense vertex, then $L^{f+1}(e) = L(e)$ (i.e. the list of available colors never changed during the execution of Line 9) and so we have $|L^{f+1}(e)| = |L(e)| \geq 2 \cdot 10^{24} \ln n > 2d_{f+1}$. □

To finish the analysis, it remains to upper bound the slacks a_i in closed form, as opposed to the (convenient but) recursive definition from Definition I.5.8. In particular, the upper bounds on a_i indicate how large the lists $L^i(e)$ need to be to make Line 9 succeed. It is clear that:

$$\begin{aligned} a_i &= 10^{24} \ln n + \sum_{j=i}^{f} \left(2\lambda_j \cdot \frac{q_j + \lambda_j}{d_j + q_j} + 16\sqrt{\lambda_j \ln n} \right) \\ &\leq 10^{24} \ln n + \sum_{j=i}^{f} \left(2\lambda_j \cdot \frac{q_j}{d_j} + 2\lambda_j \cdot \frac{\lambda_j}{d_j} + 16\sqrt{\lambda_j \ln n} \right) \\ &\leq 10^{24} \ln n + (f - i + 1) \cdot \left(2\lambda_i \cdot \frac{q_i}{d_i} + 2\lambda_i \cdot \frac{\lambda_i}{d_i} + 16\sqrt{\lambda_i \ln n} \right), \end{aligned} \tag{I.18}$$

where the last inequality follows because all three terms inside the parentheses are non-increasing in j. By the above manipulations, it remains to upper bound the quantity $f - i + 1$, where f is the number of phases. For this purpose, the following lemma is helpful:

Lemma I.5.10. *Let $a > 0$ be any number. Consider a sequence $(x_k)_{k \geq 0}$ of non-negative integer numbers, such that $x_0 \geq 1$ and:*

$$x_{k+1} := \begin{cases} x_k - \lceil a \cdot x_k^{2/3} \rceil & \text{if } x_k \geq 1, \\ 0 & \text{if } x_k < 1. \end{cases} \tag{I.19}$$

Then, for (integer) $k \geq \frac{3\sqrt[3]{x_0}}{a}$, we have $x_k \leq 1$.

Proof. We prove the statement by strong induction on the starting value $x_0 \geq 1$ of the sequence. If $x_0^{1/3} < a$, then $a x_0^{2/3} > x_k$, and in particular, $x_1 = x_0 - \lceil a \cdot x_0^{2/3} \rceil < 0$, so the statement holds.

Now assume instead that $x_0^{1/3} \geq a$, and that the statement holds for any integer $x_0' < x_0$. We apply the induction hypothesis on the sequence starting with $x_0' := x_1 = x_0 - \lceil a \cdot x_0^{2/3} \rceil$ (trivially $x_1 < x_0$). We can assume $x_1 \geq 1$, else the statement already holds. The induction hypothesis gives us that $x_{k+1} \leq 1$ for $k \geq \frac{3\sqrt[3]{x_1}}{a}$. It thus suffices to prove:

$$\frac{3\sqrt[3]{x_1}}{a} + 1 \leq \frac{3\sqrt[3]{x_0}}{a},$$

which we rearrange into:

$$\sqrt[3]{x_1} \leq \sqrt[3]{x_0} - \frac{a}{3}.$$

Now taking the third power of both sides (which are indeed both positive):

$$x_0 - \lceil a \cdot x_0^{2/3} \rceil \leq x_0 - a x_0^{2/3} + \frac{a^2 x_0^{1/3}}{3} - \frac{a^3}{9}.$$

The above inequality follows from the fact that $\lceil a x_0^{2/3} \rceil \geq a x_0^{2/3}$, and $a^2 x_0^{1/3}/3 \geq a^3/3 > a^3/9$ (since we assumed $x_0^{1/3} \geq a$). Thus the induction proof is concluded and the lemma proven. □

We are now ready to upper bound $f - i + 1$.

Lemma I.5.11. *For every $i \in \{0, \ldots, f\}$, one has $f - i + 1 \leq 7\sqrt[3]{\frac{d_i}{\ln n}}$.*

Proof. Fix such an i. Notice that for any k with $i \leq k + i \leq f$ we have $d_{k+i+1} \leq d_{k+i} - 0.9\lambda_{k+i}$ (Fact I.5.4) and $0.9\lambda_{k+i} = 0.9 \cdot d_{k+i}^{2/3} \ln^{1/3} n \geq \lceil 0.5 \cdot d_{k+i}^{2/3} \ln^{1/3} n \rceil$ for $n, d_{k+i} \geq 10$. Therefore:

$$d_{k+i+1} \leq d_{k+i} - \lceil 0.5 \cdot d_{k+i}^{2/3} \ln^{1/3} n \rceil \text{ for any } k \text{ with } i \leq k + i \leq f.$$

By Lemma I.5.10, the sequence $(d_{k+i})_{k \geq 0}$ will drop below 1 after at most $k_{\max} := 6\sqrt[3]{\frac{d_i}{\ln n}}$ steps. However, f is defined as the highest index $k + i$ for which $d_{k+i} \geq 10^{24} \ln n \geq 1$. This implies that $f \leq i + k_{\max} \leq i + 6\sqrt[3]{\frac{d_i}{\ln n}}$. In particular: $f - i + 1 \leq 6\sqrt[3]{\frac{d_i}{\ln n}} + 1 \leq 7\sqrt[3]{\frac{d_i}{\ln n}}$. □

Finally, we can now upper bound the slacks a_i:

Lemma I.5.12 (Upper bounding slacks a_i)**.** *For $d_0 \geq 1$, define the* slack sequence *$Sl(d_0)$ according to Definition I.5.8. We have, for any $i \in \{0, \ldots, f\}$:*

$$a_i \leq 10^{24} \ln n + 10^4 \cdot \left(d_i^{3/4} \ln^{1/2} d_i + d_i^{2/3} \ln^{1/3} n \right). \tag{I.20}$$

Proof. We begin by recalling (I.18):

$$a_i \leq 10^{24} \ln n + (f - i + 1) \cdot \left(2\lambda_i \cdot \frac{q_i}{d_i} + 2\lambda_i \cdot \frac{\lambda_i}{d_i} + 16\sqrt{\lambda_i \ln n} \right).$$

Replacing $\lambda_i = d_i^{2/3} \ln^{1/3} n$ and applying Lemma I.5.11, we get:

$$a_i \leq 10^{24} \ln n + 14 q_i + 14 \cdot d_i^{2/3} \ln^{1/3} n + 112 \cdot d_i^{2/3} \ln^{1/3} n.$$

Replacing $q_i = 10^2 \cdot d_i^{3/4} \ln^{1/2} d_i$ gives:

$$\begin{aligned} a_i &\leq 10^{24} \ln n + 14 \cdot 10^2 \cdot d_i^{3/4} \ln^{1/2} d_i + 126 \cdot d_i^{2/3} \ln^{1/3} n \\ &\leq 10^{24} \ln n + 10^4 \cdot \left(d_i^{3/4} \ln^{1/2} d_i + d_i^{2/3} \ln^{1/3} n \right). \end{aligned}$$

which is the claimed statement. □

Putting everything together By the above discussion, we can, with high probability, color a graph G of known (upper bound on the) maximum degree $d_0 := \Delta$ arriving online edge-by-edge assuming some mild conditions which we now discuss. In the following, we consider the *degree sequence* $D(d_0) := \{d_0, \ldots, d_{f+1}\}$ and the *slack sequence* $Sl(d_0) := \{a_0, \ldots, a_{f+1}\}$ as defined in Definition I.5.5 and Definition I.5.8.

- By Lemma I.5.7, for any phase i during the execution of Line 9, the currently uncolored subgraph U_i has maximum degree at most d_i (see Lemma I.5.7).
- For every arriving edge e of G, Line 9 expects to be provided a list $L(e)$ of available colors online. It is required that $|L(e)| \geq 2 \cdot 10^{24} \ln n$ (see Lemma I.5.9). Furthermore, the algorithm partitions (online) the set of colors $\mathcal{C} := \cup_{e \in G} L(e)$ into $C_0, \ldots, C_{f+1}$, and this partitioning is *admissible* as defined in Definition I.5.6.
- For any phase $i \in \{0, \ldots, f\}$ in which e is connected to a dense vertex of U_i, i.e., $\deg_{U_i}(v) \geq d_i - \lambda_i$, the algorithm uses the sublist $\ell^i(e) := L(e) \cap C_i(e)$, made up of currently unused colors. By the *admissible* choice of the partitioning of colors, it is guaranteed that $\lambda_i \leq |\ell^i(e)| \leq \lambda_i + 10\sqrt{\lambda_i \ln n}$.
- If, during some phase i of Line 9, the edge e is connected to a dense vertex v in U_i, i.e. $\deg_{U_i}(v) \geq d_i - \lambda_i$, the list $L^i(e)$ must have size at least $|L^i(e)| \geq d_i + a_i$. By Lemma I.5.9, it suffices that $|L^i(e)| \geq d_i + a_i$ holds for the first phase i in which e is connected to a dense vertex.

We sum up the discussion above in a concise theorem:

Theorem I.5.13 (Main Coloring Theorem)**.** *Assume the input for Line 9 is such that every arriving edge e presents a list $L(e)$ with $|L(e)| \geq 2 \cdot 10^{24} \ln n$ colors. Furthermore, the input contains a partitioning of the set of colors $\mathcal{C} := \cup_{e \in G} L(e)$ into $C_0, \ldots, C_{f+1}$, satisfying:*

1. *The partitioning $C_0, \ldots, C_{f+1}$ of the colors is* admissible *as defined in Definition I.5.6.*
2. *For any edge $e = (u, v)$, we have:*

$$|L(e)| \geq d_{i(e)} + 10^{24} \ln n + 10^4 \cdot \left(d_{i(e)}^{3/4} \ln^{1/2} d_{i(e)} + d_{i(e)}^{2/3} \ln^{1/3} n \right), \qquad \text{(I.21)}$$

where:

$$i(e) := \max\{i \in \{0, \dots, f+1\} : d_i \in D(d_0), d_i \geq \deg_G(u) \text{ and } d_i \geq \deg_G(v)\}. \tag{I.22}$$

Under the guarantee of such an input, Line 9 colors all edges e of G online with high probability (in n), such that every edge e is assigned a color $c \in L(e)$.

Proof. By the discussion preceding the theorem statement, it suffices to prove that the lists $L^i(e)$ are large enough across all phases i, such that the statement follows by Lemma I.5.7 and Lemma I.5.9. More concretely, as discussed previously and implied by Lemma I.5.9, it suffices to prove for any edge e that $|L^i(e)| \geq d_i + a_i$ holds for the first phase i in which e is connected to a dense vertex. We claim that condition 2 from the theorem statement implies this fact. Indeed, first let:

$$d_{\max}(e) := \max\{\deg_G(u) : e \in u\} \tag{I.23}$$

be the degree of the largest-degree vertex connected to e in the original graph G. Furthermore, consider $i(e)$ as defined in the theorem statement be the largest index $\leq f+1$ in the *degree sequence* of d_0 for which $d_{i(e)} \geq d_{\max}(e)$. In particular, the index of the first phase i in which e is connected to a dense vertex in U_i is at least $i(e)$. By Lemma I.5.9, if the initial list $L(e)$ provided for e has size $|L(e)| \geq d_{i(e)} + a_{i(e)}$, then for all phases $j \geq i(e)$ in Line 9 we will have $|L^j(e)| \geq d_j + a_j$. To end the proof, notice that by Lemma I.5.12, the imposed condition (I.21) implies $|L(e)| \geq d_{i(e)} + a_{i(e)}$. □

I.5.3 List Edge Coloring

In this section we consider online *list* edge coloring, where we recall that each edge $e = (u, v)$ arrives with a list $L(e) \subseteq \mathbb{N}$, and can only be colored using a color from $L(e)$ (while again guaranteeing no vertex has more than one edge of any color). Our main result for this problem is the following.

Theorem I.5.1 (Online List Edge Coloring)**.** *There exists an online list edge-coloring algorithm which, on n-vertex graphs of known maximum degree Δ, outputs with high probability (in n) a valid list-edge coloring, provided all lists $L(e)$ satisfy $|L(e)| \geq \Delta + q$, for*

$$q := 10^{24} \ln n + 10^4 \cdot \left(\Delta^{3/4} \ln^{1/2} \Delta + \Delta^{2/3} \ln^{1/3} n\right).$$

In particular, if $\Delta = \omega(\log n)$, then lists of size $\Delta + q$ with $q = o(\Delta)$ suffice.

Proof. We prove Theorem I.5.1 by running Line 9 and applying Theorem I.5.13. For this purpose, we design an online algorithm Input such that conditions 1 and 2 are fulfilled. First, by assumption, all provided lists $L(e)$ have size $|L(e)| \geq \Delta + q$, which already guarantees condition 2 of Theorem I.5.13. Hence, it remains to design an online algorithm which partitions $\mathcal{C} := \cup_{e \in G} L(e)$ into $C_0, \dots, C_{f+1}$ such

that condition 1 is fulfilled (i.e. *admissibility* of the partitioning as defined in Definition I.5.6).

As an additional point, we run Line 9 with a minor change, which can be made without loss of generality. Concretely, before the start of any phase $i \in \{0, \dots, f\}$, if any list $L^i(e)$ of remaining colors for some edge e has size $|L^i(e)| > d_i + a_i$, we prune some of the extra colors (chosen arbitrarily) in the list such that $|L^i(e)| = d_i + a_i$ holds. Effectively these extra colors are simply ignored.

Now we describe the *admissible* partitioning of colors. We define $C_0, \dots, C_f$ as follows: For any $i \in \{0, \dots, f\}$ let $\mathcal{C}^i := \mathcal{C} \setminus \cup_{0 \le j < i} C_j$ be the set of not (yet) used colors before phase i. Construct C_i by sampling (online) any color $c \in \mathcal{C}^i$ with probability $p_i := (\lambda_i + 5\sqrt{\lambda_i \ln n})/(d_i + a_i)$. We argue that p_i is a valid probability, i.e., $p_i \le 1$. Indeed, this inequality is implied by $d_i \ge \lambda_i + 5\sqrt{\lambda_i \ln n}$. To show this, notice that by Fact I.5.4, we have $5\sqrt{\lambda_i \ln n} \le 0.1\lambda_i$, such that the previous inequality is implied by $d_i \ge 1.1\lambda_i$, or $d_i \ge (1.1)^2 \ln n$ which is obviously true.

It is clear that this sampling algorithm can be implemented online. Whenever a color $c \in L(e)$ is seen for the first time, in the list $L(e)$ of some arriving edge e, we iterate through $i \in \{0, \dots, f\}$ and assign c to C_i with corresponding probability p_i (and stop the algorithm as soon as c is assigned to some such C_i). Finally, let C_{f+1} contain all those colors that have been assigned to none of $C_0, \dots, C_f$.

We claim that the above partitioning algorithm provides an *admissible* partitioning as defined in Definition I.5.6 and as required by condition 1 of Line 9. In fact, we will prove something slightly stronger by arguing the admissibility condition holds for *all* edges e, and not only those which are connected to a dense vertex. In order to do this, fix a phase $0 \le i \le f$ and assume that condition 1 was fulfilled for all previous phases and edges in the past. It remains to prove that the sublists induced by the algorithm, $\ell^i(e) := L(e) \cap C^i(e)$, have the required size for all edges e:

Lemma I.5.14. *The above partitioning algorithm induces sublists $\ell^i(e) := L^i(e) \cap C_i(e)$, which satisfy with probability at least $1 - \frac{1}{n^5}$:*

$$\lambda_i \le |\ell^i(e)| \le \lambda_i + 10\sqrt{\lambda_i \ln n} \text{ for all } e.$$

Proof of Lemma I.5.14. Fix an edge e and its list of available colors $L^i(e)$. Since condition 1 of Theorem I.5.13 is satisfied for all edges e until the current point in the execution of Line 9, we have $|L^i(e)| \ge d_i + a_i$. By our slight modification of Line 9, we may assume equality, $|L^i(e)| = d_i + a_i$.

Hence, every color $c \in L^i(e)$ is picked by the sampling algorithm with probability $p := (\lambda_i + 5\sqrt{\lambda_i \ln n})/|L^i(e)|$, such that the expected number X of chosen colors is distributed $X \sim \text{Bin}(|L^i(e)|, p)$ with $\mu := \mathbb{E}[X] = \lambda_i + 5\sqrt{\lambda_i \ln n}$. By a standard Chernoff bound, we have for any $\varepsilon \in [0, 1]$:

$$\Pr[|X - \mu| \ge \varepsilon\mu] \le 2 \exp\left(-\frac{\varepsilon^2 \cdot \lambda_i}{3}\right).$$

Fix $\varepsilon := \sqrt{18 \cdot \frac{\ln n}{\mu}}$. As $d_i \geq 10^{24} \ln n$, we have that $\mu \geq \lambda_i = d_i^{2/3} \ln^{1/3} n \geq 18 \ln n$, and so $\varepsilon \in [0, 1]$. Furthermore , we have $\varepsilon\mu \leq 5\sqrt{\lambda_i \ln n}$. Indeed, this is equivalent (by squaring) to:

$$18 \ln n \cdot (\lambda_i + 5\sqrt{\lambda_i \ln n}) \leq 25\lambda_i \ln n$$
$$90\sqrt{\lambda_i \ln n} \leq 7\lambda_i.$$

This last expression is equivalent to $d_i \geq (90/7)^4 \ln n$ which is obvious. Hence, the Chernoff inequality gives:

$$\Pr[|X - \mu| \geq 5\sqrt{\lambda_i \ln n}] \leq 2 \cdot \frac{1}{n^6} \leq \frac{1}{n^5},$$

and so $X = |\ell^i(e)| \in [\lambda_i, \lambda_i + 10\sqrt{\lambda_i \ln n}]$ with probability at least $1 - \frac{1}{n^5}$ as claimed. □

Now that the lemma is proven, it follows easily by union bound that, with high probability in n, all induced sublists $\ell^i(e)$ in all phases $i \in \{0, \ldots, f\}$ have the required size, and so condition 1 of Theorem I.5.13 is fulfilled. This finishes the proof of Theorem I.5.1. □

An improvement of Lemma I.2.1 As a final note, we observe that the slack q in Theorem I.5.1 can be naturally decomposed into two parts: the first part is the $O(\Delta^{3/4} \log^{1/2} \Delta)$-term, which comes directly from the application of Theorem I.5.3, which in turn relies on the fact that—by Theorem I.1.3—we have access to an online matching algorithm that matches any edge e with probability $1/(\Delta + \Theta(\Delta^{3/4} \log^{1/2} \Delta))$. The second part of the slack is the $O(\Delta^{2/3} \log^{1/3} n)$-term, which comes from our choice of λ in Line 9.

However, it is not hard to see that these two parts of the final slack q in Theorem I.5.1 arise independently of each other and, more generally, if one had access to an online matching algorithm with a different guarantee than $1/(\Delta + \Theta(\Delta^{3/4} \log^{1/2} \Delta))$ matching probability per edge, this would directly translate to a change in the corresponding first term of the final slack q. More concretely, for the classical online edge coloring problem, we can generalize Lemma I.2.1 to obtain the following reduction from online edge coloring to online matching:

Lemma I.5.15 (Improved Reduction)**.** *Let $\mathcal{A}$ be an online matching algorithm that, on any graph of maximum degree $\Delta = \omega(\log n)$, matches each edge with probability at least $1/(\alpha \cdot \Delta)$, for $\alpha \geq 1$. Then, there exists an online edge coloring algorithm $\mathcal{A}'$ that on any graph with maximum degree $\Delta = \omega(\log n)$ outputs an edge coloring with $(\alpha + O((\log n/\Delta)^{1/3})) \cdot \Delta$ colors with high probability in n.*

I.5.4 Local Edge Coloring

In this section we consider online *local* edge coloring, where we recall that we wish to color each edge $e = (u, v)$ with a color not much higher than $d_{\max}(e) := \max\{\deg(u), \deg(v)\}$. Our main result for this problem is the following.

Theorem I.5.2 (Online Local Edge Coloring)**.** *There exists an online edge-coloring algorithm which, on n-vertex graphs with* a priori *known degree sequence* $\{\deg(v) \mid v \in V\}$,[5] *computes, with high probability (in n), an edge coloring $c : E \to \mathbb{N}$ that colors each edge e using a color $c(e)$ which satisfies:*

$$c(e) \leq d_{max}(e) + 2 \cdot 10^{24} \ln n + 10^5 \cdot \left(d_{max}^{3/4}(e) \ln^{1/2} d_{max}(e) + d_{max}^{2/3}(e) \ln^{1/3} n \right).$$

In particular, if $d_{max}(e) = \omega(\log n)$, then $c(e) \leq d_{\max}(e) \cdot (1 + o(1))$.

Proof. The statement of Theorem I.5.2 is almost an immediate consequence of our more general Theorem I.5.13. For $i \in \{0, \ldots, f+1\}$, let d_i and a_i denote the entries from the *degree sequence* and *slack sequence* of $d_0 := \Delta(G)$ as defined in Definition I.5.5 and Definition I.5.8 (also see Theorem I.5.13). We define the set of colors $\mathcal{C}$ to be $\mathcal{C} := \{1, \ldots, d_0 + a_0\}$ and propose the following partitioning:

$$C^i := \{d_i + a_i - (\lambda_i - 1), \ldots, d_i + a_i\} \text{ for } i \in \{0, \ldots, f\}$$
$$C^{f+1} := \{1, \ldots, 2 \cdot d_f\}.$$

The fact that this is a valid partitioning follows from Lemma I.5.9.

Now fix an edge $e = (u, v)$ and let $d_{\max}(e) := \max\{\deg(u), \deg(v)\}$. If $d_{\max}(e) \leq d_{f+1}$, consider the following input list $L(e)$ of available colors:

$$L(e) := \{1, \ldots, 2 \cdot d_{f+1}\}. \tag{I.24}$$

As e is never connected to a dense vertex during phases $i \in \{0, \ldots, f\}$, condition 1 is vacuously true. Furthermore, e can be assigned in phase $f+1$ a color $c(e) \in \ell^{f+1}(e) = L(e) \cap C^{f+1} = L(e)$, such that $c(e) \leq 2 \cdot d_{f+1} \leq 2 \cdot 10^{24} \ln n$, which implies the statement from Theorem I.5.13 for edge e.

In the following, assume $d_{\max}(e) > d_{f+1}$. Define $i(e)$ as in Theorem I.5.13 and notice that $i(e) \leq f$ because $d_{\max}(e) > d_{f+1}$. Consider the following input list $L(e)$:

$$L(e) := \left\{1, \ldots, d_{i(e)} + 10^{24} \ln n + 10^4 \cdot \left(d_{i(e)}^{3/4} \ln^{1/2} d_{i(e)} + d_{i(e)}^{2/3} \ln^{1/3} n \right) \right\}. \tag{I.25}$$

Condition 2 of Theorem I.5.13 is trivially fulfilled. Consider the induced partitioning of $L(e)$ into sublists $\ell^0(e), \ldots, \ell^{f+1}(e)$, where $\ell^i(e) := L(e) \cap C^i$. It holds that $|\ell^i(e)| = \lambda_i$ for any phase $i \leq f$ in which edge e is connected to a dense vertex of U_i.

[5] This theorem also holds if $\deg(v)$ are only *upper bounds* for the true degrees, in which case the guarantees of the theorem will be with respect to these upper bounds.

This is because, for all such i, we have $C^i \subseteq L(e)$ by Lemma I.5.12, and $|C^i| = \lambda_i$. This property guarantees condition 1 of Theorem I.5.13.

By Theorem I.5.13, we obtain that, with high probability, Line 9 assigns each edge e a color $c(e)$ satisfying $c(e) \leq |L(e)|$. Recall $d_{\max}(e) := \max\{\deg(u), \deg(v)\} > d_{f+1}$, which implies $i(e) \leq f$. First, we will make use the following inequality, whose proof is deferred:

Lemma I.5.16. *With the above notations, and assuming $i(e) \leq f$, it holds that:*

$$d_{i(e)} \leq d(e) := d_{max}(e) + 2 \cdot d_{max}^{2/3}(e) \ln^{1/3} n. \tag{I.26}$$

Combining with the definition of $L(e)$ in (I.25) and with the fact that $c(e) \leq |L(e)|$, we obtain by the above lemma that:

$$c(e) \leq |L(e)| \leq g(e) := d(e) + 10^{24} \ln n + 10^4 \cdot \left(d(e)^{3/4} \ln^{1/2} d(e) + d(e)^{2/3} \ln^{1/3} n \right). \tag{I.27}$$

To finish the proof of the theorem, it suffices to show that:

Lemma I.5.17. *With $g(e)$ as defined above, we have:*

$$g(e) \leq d_{max}(e) + 2 \cdot 10^{24} \ln n + 10^5 \cdot \left(d_{max}^{3/4}(e) \ln^{1/2} d_{max}(e) + d_{max}^{2/3}(e) \ln^{1/3} n \right). \tag{I.28}$$

As in the case of Lemma I.5.16, we defer the proof (see below). By combining (I.27) and (I.28), we get the desired upper bound for $c(e)$, and the statement of Theorem I.5.2 follows. □

We now present the proofs of lemmas Lemma I.5.16 and Lemma I.5.17, which were deferred previously:

Proof of Lemma I.5.16. By definition, $i(e)$ is the largest index $i \in \{0, \ldots, f+1\}$, such that $d_{i(e)} \geq d_{\max}(e)$. As $i(e) \neq f+1$, it follows that $d_{i(e)+1} \leq d_{\max}(e)$, and from Definition I.5.5 we have $d_{i(e)+1} \geq d_{i(e)} - \lambda_{i(e)}$. Recalling that $\lambda_{i(e)} = d_{i(e)}^{2/3} \ln^{1/3} n$, we conclude:

$$d_{\max}(e) \geq d_{i(e)} - \lambda_{i(e)} = d_{i(e)} \cdot \left(1 - \sqrt[3]{\frac{\ln n}{d_{i(e)}}} \right) \geq d_{i(e)} \cdot \left(1 - \sqrt[3]{\frac{\ln n}{d_{\max}(e)}} \right). \tag{I.29}$$

Notice that, as $i(e) \leq f$, we have $d_{i(e)} \geq 10^{24} \ln n$, which implies $d_{\max}(e) \geq d_{i(e)} - \lambda_{i(e)} > 4 \ln n$, such that $\sqrt[3]{\frac{\ln n}{d_{\max}(e)}} < 1/2$. Therefore, inequality (I.29) gives:

$$\begin{aligned} d_{i(e)} &\leq d_{\max}(e) \cdot \frac{1}{1 - \sqrt[3]{\frac{\ln n}{d_{\max}(e)}}} \\ &\leq d_{\max}(e) \cdot \left(1 + 2 \cdot \sqrt[3]{\frac{\ln n}{d_{\max}(e)}} \right) = d_{\max(e)} + 2 \cdot d_{\max}^{2/3}(e) \ln^{1/3} n. \end{aligned}$$

□

Proof of Lemma I.5.17. Recall that $g(e)$ was defined as:

$$g(e) := d(e) + 10^{24} \ln n + 10^4 \cdot \left(d(e)^{3/4} \ln^{1/2} d(e) + d(e)^{2/3} \ln^{1/3} n \right), \tag{I.30}$$

where $d(e) = d_{\max}(e) + 2 \cdot d_{\max}^{2/3}(e) \ln^{1/3} n$. Now, if $d_{\max}(e) \leq \ln n$, then $d(e) \leq 3 \ln n$. But then (I.30) immediately gives that $g(e) \leq 2 \cdot 10^{24} \ln n$, which implies the lemma. Conversely, if $d_{\max}(e) \geq \ln n$, then $d(e) \leq 3 \cdot d_{\max}(e)$. Therefore, we have:

$$d(e)^{3/4} \ln^{1/2} d(e) \leq 10 \cdot d_{\max}^{3/4}(e) \ln^{1/2} d_{\max}(e)$$
$$d(e)^{2/3} \ln^{1/3} n \leq 3 \cdot d_{\max}^{2/3}(e) \ln^{1/3} n.$$

But then, by (I.30), we obtain the desired inequality:

$$\begin{aligned} g(e) &\leq d_{\max}(e) + 2 \cdot d_{\max}^{2/3}(e) \ln^{1/3} n + 10^{24} \ln n + \\ &\qquad 10^4 \cdot \left(10 \cdot d_{\max}^{3/4}(e) \ln^{1/2} d_{\max}(e) + 3 \cdot d_{\max}^{2/3}(e) \ln^{1/3} n \right) \\ &\leq d_{\max}(e) + 10^{24} \ln n + 10^5 \cdot \left(d_{\max}^{3/4}(e) \ln^{1/2} d_{\max}(e) + d_{\max}^{2/3}(e) \ln^{1/3} n \right). \end{aligned}$$

□

I.6 Extension: Online Rounding of Fractional Matchings

In this section we generalize the result in Section I.4 to a rounding algorithm for fractional matchings. By *fractional matching*, we mean an assignment $x \in [0,1]^E$ to the edges such that for each vertex v, we have $\sum_{e \in \delta(v))} x_e \leq 1$. In the online model, the value of x_e is revealed alongside the edge e. An online rounding algorithm must decide immediately and irrevocably whether to match the newest arriving edge. The objective is to match each edge e with probability close to x_e.

Algorithm I.2 can be viewed as an online algorithm which rounds the uniform fractional matching $\{x_e = \frac{1}{\Delta}\}_{e \in E}$ to an integral one, while nearly preserving marginals, $\Pr[e \text{ matched}] = \frac{1}{\Delta}(1 - o(1))$. We show that one can—in a straightforward manner—generalize our Algorithm I.2 to round arbitrary fractional matchings (also in non-bipartite graphs, see Remark I.6.2), as long as they are sufficiently "spread out", resulting in our Algorithm I.5 and Theorem I.6.1. In particular, as long as all $x_e \leq \varepsilon$, we will be able to round $\vec{x}$ while only incurring some a small loss which goes to zero when $\varepsilon \to 0$. To our knowledge, this is the first online rounding algorithm with such guarantees for general (non-bipartite) graphs, and also for adversarial edge arrivals in any (bipartite) graph.

Theorem I.6.1. *Let $x \in \mathbb{R}^E$ be a fractional matching of some graph G, which is revealed online, and satisfies $x_e \leq \varepsilon$ for all edges e, for some known $\varepsilon \leq 0.99$. Then there exists a randomized online matching algorithm whose output matching $\mathcal{M}$ satisfies for any edge e:*

$$x_e \geq \Pr[e \text{ matched by } \mathcal{M}] \geq x_e \cdot (1 - s(\varepsilon)),$$

Algorithm I.5: MatchingAlgorithmChanges

1 *At the arrival of edge* $e_t = (u, v)$ *at time* t*:* Sample $X_t \sim [0, 1]$ uniformly at random. Define

$$P(e_t) = \begin{cases} x_e \cdot (1 - s(\varepsilon)) \cdot \frac{1}{F_t(u) \cdot F_t(v)} & \text{if } u \text{ and } v \text{ are unmatched in } M_t, \\ 0 & \text{otherwise.} \end{cases}$$

2 and

$$\hat{P}(e_t) = \begin{cases} P(e_t) & \text{if } \min\{F_t(u), F_t(v)\} \cdot (1 - P(e_t)) \geq \frac{s(\varepsilon)}{4} \\ 0 & \text{otherwise.} \end{cases}$$

3 Set

- $F_{t+1}(u) \leftarrow F_t(u) \cdot (1 - \hat{P}(e_t))$;
- $F_{t+1}(v) \leftarrow F_t(v) \cdot (1 - \hat{P}(e_t))$;
- $M_{t+1} = \begin{cases} M_t \cup \{e_t\} & \text{if } X_t < \hat{P}(e_t), \\ M_t & \text{otherwise.} \end{cases}$

where $s(\varepsilon) := C \cdot \sqrt[4]{\varepsilon} \cdot \sqrt[2]{\left(\ln \frac{1}{\varepsilon}\right)}$ *(for some large enough constant* $C > 0$*) converges to* 0 *as* $\varepsilon \to 0$.

Remark I.6.2. Note that in general graphs, fractional matchings can be $3/2$ times larger than their largest integral counterparts, as exemplified by a triangle with values $x_e = 1/2$ for all its edges. That is, the integrality gap of this relaxation of matchings is $3/2$. Nonetheless, our algorithm works for non-bipartite graphs, and for sufficiently spread out fractional matchings we almost losslessly round them to integral matchings. This does not contradict the integrality gap of this relaxation in general graphs, as all "odd set" constraints in the integral matching polytope [Edm65] are approximately satisfied for spread out fractional matching $x_e \leq \varepsilon$. On the other hand, to round fractional matchings in non-bipartite graphs, it is necessary to incur some loss (with respect to ε) when rounding, even in offline settings.

The new algorithm, a straightforward adaptation of Algorithm I.2, is given by Algorithm I.5. The following results are proven analogously to their counterparts from Section I.4, and are therefore omitted here:

Observation I.6.3 (Corresponds to Observation I.4.3)**.** $F_t(v) \geq \frac{s(\varepsilon)}{4}$ *and* $\hat{P}(e_t) \leq P(e_t) \leq \frac{\varepsilon}{s^2(\varepsilon)}$ *for every vertex* $v \in V$ *and time* t.

Observation I.6.4 (Corresponds to Observation I.4.1)**.** *For any t, the random variables $F_t(v), P(e_t), \hat{P}(e_t)$ are determined by the current partial input $e_1, \ldots, e_t$ and the current matching M_{t-1}.*

Lemma I.6.5 (Corresponds to Lemma I.4.2)**.** *For any edge e_t it holds that $\Pr[X_t < P(e_t)] = x_e \cdot (1 - s(\varepsilon))$.*

Lemma I.6.6 (Corresponds to Observation I.4.4)**.** *If $e_t = (u, v)$ and $\min\{F_t(u),\ F_t(v)\} \geq s(\varepsilon)/3$, then $\hat{P}(e_t) = P(e_t)$.*

The proof of Lemma I.6.6 (just as its counterpart Observation I.4.4) requires $P(e_t) \leq \frac{\varepsilon}{s^2(\varepsilon)} \leq 1/4$. This can be achieved by imposing:

$$C \geq \max_{\varepsilon} \left(\frac{4\varepsilon}{\sqrt{\varepsilon} \cdot \ln \frac{1}{\varepsilon}} \right). \tag{I.31}$$

However, as C is supposed to be a constant, we still need to check that the right hand side of the above inequality is also a constant. By the statement of Theorem I.6.1, we have that $\varepsilon \leq 0.99$, and it is easy to check that the function $\varepsilon \mapsto \left(\frac{\varepsilon}{\sqrt{\varepsilon} \cdot \ln \frac{1}{\varepsilon}} \right)$ is bounded in the interval $(0, 0.99)$. Hence, the right hand side of (I.31) is a constant.

The analogue of Lemma I.4.5 is:

Lemma I.6.7 (Corresponds to Lemma I.4.5)**.** *Let $e_{t_1} = (u_1, v), \ldots, e_{t_\ell} = (u_\ell, v)$ be the edges incident to v with $t_1 < \cdots < t_\ell$. Further, let $S := \{u_i \mid u_i \notin M_{t_i}\}$ be those neighbors u_i that are unmatched by time t_i when edge $e_{t_i} = (u_i, v)$ arrives. If:*

$$\sum_{u_i \in S} \frac{x_{e_{t_i}} \cdot (1 - s(\varepsilon))}{F_{t_i}(u_i)} \leq 1 - \frac{s(\varepsilon)}{3}, \tag{I.32}$$

then $F(v) \geq s(\varepsilon)/3$.

For ease of notation let $x_i := x_{e_{t_i}}$ be the fractional input of the edge $e_{t_i} = (u_i, v)$, which connects v to its neighbor u_i. We will derive a martingale from the following quantities:

$$S_t := \{u_i \in N(v) \mid u_i \notin M_{\min\{t,t_i\}}\} \quad \text{and} \quad Y_{t-1} := \sum_{u_i \in S_t} \frac{x_i \cdot (1 - s(\varepsilon))}{F_{\min\{t,t_i\}}(u_i)}.$$

Claim I.6.8 (Corresponds to Lemma I.4.6)**.** *$Y_0, \ldots, Y_m$ form a martingale w.r.t. the random variables $X_1, \ldots, X_m$. Furthermore, the difference $Y_t - Y_{t-1}$ is given by the following two cases:*

- *If e_t is added to M_{t+1}, which happens with probability $\hat{P}(e_t)$, then:*

$$Y_t - Y_{t-1} = -\sum_{u_i \in S_t \cap e_t} \frac{x_i \cdot (1 - s(\varepsilon))}{F_t(u_i)}. \tag{I.33}$$

- *If instead e_t is not added to M_{t+1}, which happens with probability $1-\hat{P}(e_t)$, then:*

$$Y_t - Y_{t-1} = \frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \cdot \sum_{u_i \in S_t \cap e_t} \frac{x_i \cdot (1-s(\varepsilon))}{F_t(u_i)}. \tag{I.34}$$

To apply Freedman's inequality, as in Section I.4, bounds on the step size and on the variance of the martingale are required. They are given by the following two lemmas:

Lemma I.6.9 (Corresponds to Lemma I.4.8)**.** *For all times t and realization of the randomness, $|Y_t - Y_{t-1}| \leq A$, where $A := \frac{8\varepsilon}{s(\varepsilon)}$.*

Proof. By using the expressions for the difference $Y_t - Y_{t-1}$ from Lemma I.6.7, we obtain:

$$|Y_t - Y_{t-1}| \leq \max\left\{\frac{\hat{P}(e_t)}{1-\hat{P}(e_t)}, 1\right\} \cdot \sum_{u_i \in S_t \cap e_t} \frac{x_i(1-s(\varepsilon))}{F_t(u_i)} \leq \sum_{u_i \in S_t \cap e_t} \frac{\varepsilon}{s(\varepsilon)/4} \leq \frac{8\varepsilon}{s(\varepsilon)}.$$

For the second inequality, first notice that we already guarantee $\hat{P}(e_t) \leq P(e_t) \leq 1/4$ for the proof of Lemma I.6.6, so in particular $\frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \leq 1$. Also, we have $F_t(u_i) \geq s(\varepsilon)/4$ (by Observation I.6.4) at any point of time in the algorithm. For the third inequality we used the trivial fact that $|S_t \cap e_t| \leq 2$. □

Lemma I.6.10 (Corresponds to Lemma I.4.9)**.** *Consider the martingale described above. We have:*

$$\sum_{t=1}^{m} \mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_{t-1}, \ldots X_1] \leq 128 \ln\left(\frac{4}{s(\varepsilon)}\right) \cdot \frac{\varepsilon}{(s(\varepsilon))^2}. \tag{I.35}$$

Proof. Mimicking the proof of Lemma I.4.9: Assuming edge e_t arrives at time t, we have:

$$\begin{aligned}
\mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_{t-1}, \ldots X_1] \leq \\
2\hat{P}(e_t) \cdot \sum_{u_i \in S_t \cap e_t} \left(\frac{x_i \cdot (1-s(\varepsilon))}{F_t(u_i)}\right)^2 + \\
2(1-\hat{P}(e_t)) \sum_{u_i \in S_t \cap e_t} \cdot \left(\frac{\hat{P}(e_t) \cdot x_i \cdot (1-s(\varepsilon))}{F_t(u_i)(1-\hat{P}(e_t))}\right)^2 = \\
\sum_{u_i \in S_t \cap e_t} \frac{2\hat{P}(e_t) \cdot x_i^2 \cdot (1-s(\varepsilon))^2}{(F_t(u_i))^2} \left(1 + \frac{\hat{P}(e_t)}{1-\hat{P}(e_t)}\right) \leq \\
128 \frac{\hat{P}(e_t)}{(s(\varepsilon))^2} \cdot x_i^2,
\end{aligned}$$

where we used $F_t(u_i) \geq s(\varepsilon)/4$, $1 + \frac{\hat{P}(e_t)}{1-\hat{P}(e_t)} \leq 2$ and $|S_t \cap e_t| \leq 2$.

We sum the above inequality over all t. An edge e_t is thereby summed over on the right hand side only if e_t is incident to some vertex $u_i \in S_t$. As $S_t \subseteq S_0 = N(v)$, we have the following upper bound:

$$\sum_{t=1}^{m} \mathbb{E}[(Y_t - Y_{t-1})^2 \mid X_{t-1}, \dots X_1] \leq \sum_{i=1}^{\ell} x_i^2 \cdot \sum_{e_t \in \delta(u_i)} 128 \frac{\hat{P}(e_t)}{(s(\varepsilon))^2}. \tag{I.36}$$

For any neighbor u_i of v we have:

$$s(\varepsilon)/4 \leq F_m(u_i) = \prod_{e_t \in \delta(u_i)} (1 - \hat{P}(e_t)) \leq \exp\left(- \sum_{e_t \in \delta(u_i)} \hat{P}(e_t) \right),$$

which implies that:

$$\sum_{e_t \in \delta(u_i)} \hat{P}(e_t) \leq \ln\left(\frac{4}{s(\varepsilon)}\right).$$

Hence:

$$\sum_{i=1}^{\ell} x_i^2 \cdot \sum_{e_t \in \delta(u_i)} 128 \frac{\hat{P}(e_t)}{(s(\varepsilon))^2} \leq \sum_{i=1}^{\ell} x_i^2 \cdot 128 \ln\left(\frac{4}{s(\varepsilon)}\right) \cdot \frac{1}{(s(\varepsilon))^2} \leq 128 \ln\left(\frac{4}{s(\varepsilon)}\right) \cdot \frac{\varepsilon}{(s(\varepsilon))^2},$$

because $\sum_{i=1}^{\ell} x_i^2 = \sum_{e:=\{v,u_i\}} x_e^2 \leq \varepsilon \cdot \sum_{e:=\{v,u_i\}} x_e \leq \varepsilon$. □

We are finally ready to analyze our online rounding algorithm.

Proof of Theorem I.6.1. First, we remark that the proof of the upper bound is straightforward, as we have:

$$\Pr[e_t \text{ matched}] = \Pr[X_t < \hat{P}(e_t)] \leq \Pr[X_t < P(e_t)] = x_e \cdot (1 - s(\varepsilon)) \leq x_e,$$

where the equality $\Pr[X_t < P(e_t)] = x_e \cdot (1 - s(\varepsilon))$ follows by Lemma I.6.5.

On the other hand, $\Pr[e_t \text{ matched}] = \Pr[X_t < \hat{P}(e_t)]$ can be expanded as follows:

$$\begin{aligned}\Pr[e_t \text{ matched}] = &\Pr[X_t < P(e_t)] - \\ &\Pr[X_t < P(e_t) \mid \hat{P}(e_t) \neq P(e_t)] \cdot \Pr[\hat{P}(e_t) \neq P(e_t)].\end{aligned}$$

By Lemma I.6.5, we have that $\Pr[X_t < P(e_t)] = x_e \cdot (1 - s(\varepsilon))$. Moreover, using the definition of $P(e_t)$ and the fact that $F_t(u), F_t(v) \geq s(\varepsilon)/4$ (Observation I.6.3), we have that, deterministically:

$$P(e_t) = x_e \cdot (1 - s(\varepsilon)) \cdot \frac{1}{F_t(u) \cdot F_t(v)} \leq x_e \cdot \frac{16}{s^2(\varepsilon)}.$$

Consequently, $\Pr[X_t < P(e_t) \mid \hat{P}(e_t) \neq P(e_t)] \leq x_e \cdot \frac{16}{s^2(\varepsilon)}$. Finally, we leverage our martingale to upper bound the probability of $\hat{P}(e_t) \neq P(e_t)$, as follows. Fix an endpoint $v \in e_t$, and consider the martingale Y associated with $F_t(v)$. First, we note that $Y_0 := \sum_{u_i \in S_t} x_e \cdot (1 - s(\varepsilon)) \leq 1 - s(\varepsilon)$. By Lemmas I.6.6 and I.6.7, we want to prove that $Y_m \leq 1 - s(\varepsilon)/3$ is unlikely, and so it is sufficient to upper bound $\Pr[|Y_0 - Y_m| \geq 2s(\varepsilon)/3]$. By Freedman's inequality, and taking $s(\varepsilon) = C \cdot \sqrt[4]{\varepsilon} \cdot \sqrt[2]{\left(\ln \frac{1}{\varepsilon}\right)}$ (for some sufficiently large constant C), we obtain:

$$\Pr[|Y_0 - Y_m| \geq 2s(\varepsilon)/3] \leq 2\exp\left(-\frac{\left(\frac{2}{3}s(\varepsilon)\right)^2}{2\left(128\ln\left(\frac{4}{s(\varepsilon)}\right) \cdot \frac{\varepsilon}{(s(\varepsilon))^2}\right) + \frac{8\varepsilon}{s(\varepsilon)} \cdot \left(\frac{2}{3}s(\varepsilon)\right)}\right) \leq 2\varepsilon^5.$$

Consequently, by union bounding over both endpoints of e_t, we have that:

$$\Pr[\hat{P}(e_t) \neq P(e_t)] \leq 4\varepsilon^5.$$

Combining the above into our expanded form for $\Pr[e_t \text{ matched}]$, we obtain the desired inequality:

$$\Pr[e_t \text{ matched}] \geq x_e \cdot (1 - s(\varepsilon)) - x_e \cdot \frac{16}{s^2(\varepsilon)} \cdot 4\varepsilon^5 \geq x_e \cdot (1 - 2s(\varepsilon)),$$

where the last inequality follows because it is equivalent to $C^3 \cdot \varepsilon^{3/4} \left(\ln \frac{1}{\varepsilon}\right)^{3/2} - 64 \cdot \varepsilon^5 \geq 0$. This can be verified directly for $C \geq 40$ and $0 < \varepsilon \leq 0.99$. □

I.6.1 Application: Online Matching with Unknown Δ

In this section, we point out an immediate application of our generalized fractional matching rounding algorithm (Theorem I.6.1 and Algorithm I.5). Here we consider the online edge coloring problem, but where the maximum degree Δ is initially unknown. Again, using $2\Delta - 1$ colors is easy by a greedy algorithm which always colors an edge with the smallest available color. In contrast to the case of known Δ, it is proven that a 1.606-competitive algorithm is the best we can hope for when it comes to unknown Δ, even in the case of large $\Delta = \omega(\log n)$ and vertex arrivals [CPW19]. We approach this lower bound, and in so doing obtain the first online algorithm beating the naive $(2\Delta - 1)$-edge-coloring algorithm for unknown Δ, under general vertex arrivals.

Given our rounding scheme of Theorem I.6.1, our result for unknown Δ follows directly from known reductions introduced in [CPW19] (see also the end of [Waj20, Chapter 6]). For completeness, we provide a sketch including pointers to the relevant ingredients in [CPW19].

Theorem I.6.11. *There exists an online algorithm which on n-vertex general graphs with maximum degree Δ only known to satisfy a lower bound $\Delta \geq \Delta' = \omega(\log n)$,*

with the graph revealed vertex-by-vertex, computes a $(1.777 + o(1)) \cdot \Delta$*-edge-coloring with high probability.*

Proof (Sketch). [CPW19] study the following relaxation of edge coloring; in this relaxation a fractional $\alpha\Delta$-edge-coloring consists of $\alpha\Delta$ many fractional matchings such that each edge is matched to an extent of (at least) one when summed across all fractional matchings. In their terminology, a graph is shown to be *fractionally* k*-edge-colorable* if the following LP has a solution.

$$\begin{aligned} \sum_{c \in [k]} x_{e,c} &= 1 && \forall e \in E \\ \sum_{e \ni v} x_{e,c} &\leq 1 && \forall v \in V, c \in [k] \\ x_{e,c} &\geq 0 && \forall e \in E, c \in [k] \end{aligned}$$

For graphs with unknown degree, [CPW19] provide online fractional edge coloring algorithms using $e/(e-1)\Delta$ and 1.777Δ matchings under one-sided vertex arrivals in bipartite graphs and arbitrary vertex arrivals in general graphs, respectively. Both factional algorithms maintain collections of fractional matchings $\{x_{e,c}\}_c$ with bounded ℓ_∞ norm, $\max_e x_{e,c} = o(1)$, whenever $\Delta = \omega(1)$. [CPW19] further provide (see their Algorithm 2) a rounding framework to round these; they show how to convert online fractional $\alpha\Delta$-edge-coloring algorithms with the above bounded ℓ_∞ norm guarantee into (randomized) online $(\alpha + o(1))\Delta$-edge-coloring algorithms for graphs with unknown maximum degree Δ satisfying $\Delta \geq \Delta' = \omega(\log n)$, with Δ' known. (The above $o(1)$ term is of the form $\text{poly}(\log n/\Delta')$.) An important ingredient for their framework is a $(1 + o(1))$-approximate rounding scheme for online fractional matchings $\vec{x}$ with $\max_e x_e = o(1)$. Such a rounding scheme for fractional matchings under one-sided vertex arrivals in bipartite graphs was given by [CW18] (see [Waj20, Chapter 5]). Our Theorem I.6.1 provides such a rounding scheme under edge arrivals (and hence also under vertex arrivals) in general graphs. Combining our new rounding scheme with the rounding framework of [CPW19] then allows us to round their online fractional 1.777Δ-edge-coloring algorithm and obtain an online $(1.777 + o(1))\Delta$-edge-coloring algorithm under general vertex arrivals, as claimed. □

Bibliography

[AGKM11] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. "Online vertex-weighted bipartite matching and single-bid budgeted allocations". In: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2011, pp. 1253–1264 (cit. on p. 416).

[AMSZ03] Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. "Switch scheduling via randomized edge coloring". In: *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*. 2003, pp. 502–512 (cit. on pp. 413, 417).

[BBKTW94] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. "On the power of randomization in on-line algorithms". In: *Algorithmica* 11.1 (1994), pp. 2–14 (cit. on p. 417).

[BC21] Guy Blanc and Moses Charikar. "Multiway Online Correlated Selection". In: *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*. 2021, pp. 1277–1284 (cit. on p. 416).

[BDG16] Nikhil Bansal, Daniel Dadush, and Shashwat Garg. "An Algorithm for Komlós Conjecture Matching Banaszczyk's bound". In: *SIAM Journal on Computing* 48 (2016) (cit. on p. 419).

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. "Online Edge Coloring Algorithms via the Nibble Method". In: *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2021, pp. 2830–2842 (cit. on pp. 413, 417).

[BMM12] Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. "Online Graph Edge-Coloring in the Random-Order Arrival Model." In: *Theory of Computing* 8.1 (2012), pp. 567–595 (cit. on pp. 413, 417).

[BMN92] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. "The greedy algorithm is optimal for on-line edge coloring". In: *Information Processing Letters (IPL)* 44.5 (1992), pp. 251–253 (cit. on pp. 413, 416, 417).

[BSVW24] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. "Simple and Optimal Online Bipartite Edge Coloring". In: *Proceedings of the 7th Symposium on Simplicity in Algorithms (SOSA)*. 2024 (cit. on pp. 413, 417).

[BSVW25] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. "Deterministic Online Bipartite Edge Coloring". In: *arXiv preprint arXiv:2408.03661* (2025). to appear in SODA 2025 (cit. on p. 417).

[Chr23] Aleksander Bjørn Grodt Christiansen. "The power of multi-step Vizing chains". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 1013–1026 (cit. on p. 415).

[CPW19] Ilan Reuven Cohen, Binghui Peng, and David Wajc. "Tight Bounds for Online Edge Coloring". In: *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 1–25 (cit. on pp. 413–418, 436, 456, 457).

[CW18] Ilan Reuven Cohen and David Wajc. "Randomized Online Matching in Regular Graphs". In: *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2018, pp. 960–979 (cit. on pp. 414, 415, 436, 457).

[DGS24] Aditi Dudeja, Rashmika Goswami, and Michael Saks. "Randomized Greedy Online Edge Coloring Succeeds for Dense and Randomly-Ordered Graphs". In: *arXiv preprint arXiv:2406.13000* (2024) (cit. on p. 417).

[DR96] Devdatt Dubhashi and Desh Ranjan. "Balls and bins: A study in negative dependence". In: *BRICS Report Series* 3.25 (1996) (cit. on p. 438).

[Edm65] Jack Edmonds. "Maximum matching and a polyhedron with 0, 1-vertices". In: *Journal of research of the National Bureau of Standards B* 69.125-130 (1965), pp. 55–56 (cit. on p. 452).

[FHTZ20] Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. "Edge-Weighted Online Bipartite Matching". In: *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 412–423 (cit. on p. 416).

[Fre75] David A Freedman. "On Tail Probabilities for Martingales". In: *The Annals of Probability* (1975) (cit. on p. 419).

[GHHNYZ21] Ruiquan Gao, Zhongtian He, Zhiyi Huang, Zipei Nie, Bijun Yuan, and Yan Zhong. "Improved Online Correlated Selection". In: *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*. 2021, pp. 1265–1276 (cit. on p. 416).

[GKMSW19] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. "Online matching with general arrivals". In: *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 26–38 (cit. on p. 416).

[HKTWZZ20] Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. "Fully Online Matching". In: *Journal of the ACM (JACM)* 67.3 (2020), pp. 1–25 (cit. on p. 416).

[HMRR98] Michel Habib, Colin McDiarmid, Jorge Ramirez-Alfonsin, and Bruce Reed. *Probabilistic methods for algorithmic discrete mathematics*. Vol. 16. Springer Science & Business Media, 1998 (cit. on p. 419).

[HTWZ20] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. "Fully Online Matching II: Beating Ranking and Water-filling". In: *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*. 2020, To appear (cit. on p. 416).

[HZZ20] Zhiyi Huang, Qiankun Zhang, and Yuhao Zhang. "AdWords in a Panorama". In: *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*. 2020, To appear (cit. on p. 416).

[Kah96] Jeff Kahn. "Asymptotically good list-colorings". In: *Journal of Combinatorial Theory, Series A* 73.1 (1996), pp. 1–59 (cit. on p. 415).

[KLSST22] Janardhan Kulkarni, Yang P Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. "Online Edge Coloring via Tree Recurrences and Correlation Decay". In: *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*. 2022, pp. 2958–2977 (cit. on pp. 413, 415, 417).

[KVV90] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. "An optimal algorithm for on-line bipartite matching". In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*. 1990, pp. 352–358 (cit. on p. 416).

[MSVV07] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. "Adwords and generalized online matching". In: *Journal of the ACM (JACM)* 54.5 (2007), p. 22 (cit. on p. 416).

[NSW23] Joseph (Seffi) Naor, Aravind Srinivasan, and David Wajc. "Online Dependent Rounding Schemes". In: *arXiv preprint arXiv:2301.08680* (2023) (cit. on p. 416).

[SW21] Amin Saberi and David Wajc. "The Greedy Algorithm is *not* Optimal for On-Line Edge Coloring". In: *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP)*. 2021, 109:1–109:18 (cit. on pp. 413, 417, 418).

[Viz64] Vadim G Vizing. "On an estimate of the chromatic class of a p-graph". In: *Diskret analiz* 3 (1964), pp. 25–30 (cit. on pp. 413, 418).

[Viz76] Vadim G Vizing. "Vertex colorings with given colors". In: *Diskret. Analiz* 29 (1976), pp. 3–10 (cit. on p. 415).

[Waj17] David Wajc. *Negative association – definition, properties, and applications*. `http://www.cs.cmu.edu/dÌČwajc/notes/NegativeAssociation.pdf`. 2017 (cit. on p. 438).

[Waj20] David Wajc. "Matching Theory Under Uncertainty". PhD thesis. Carnegie Mellon University, 2020 (cit. on pp. 416, 436, 438, 456, 457).