Algorithms for Fundamental Problems in Computer Networks

by

Hsin-Hao Su

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2015

Doctoral Committee:

Associate Professor Seth Pettie, Chair

Professor Anna Gilbert

Assistant Professor Grant Schoenebeck

Professor Roger Wattenhofer, ETH Zürich

Acknowledgements

Foremost, I would like to deeply thank my advisor, Seth Pettie, for his continous support of my Ph.D. study. Without his patience and guidance, I would not know how to do research in a proper way and this thesis would not be possible.

I must also thank my committee members, Anna Gilbert, Grant Schoenebeck, Roger Wattenhofer for their reviews and helpful comments on my thesis. I was my great honor to have you serving on my committee.

I went with Seth to Denmark during his sabbatical (an idea originated from Yaoyun Shi). It was one of the best years of my life to live in the world's happiest country. While I was there, I took side-trips to visit Michael Elkin at Ben-Gurion University and Roger Wattenhofer at ETH. I really appreciate their generous and warm hospitality. I am in debted to Lars Arge, and all the members at the MADALGO, who provided me with a productive working environment. I never expected that I could travel around the world during my Ph.D. study. Some suggested that I should start running a travel agency if I decided to change my career.

I also like to thank my collaborators and friends, from whom I have learned a lot from and who made my Ph.D. life more enjoyable. The list includes Kai-Min Chung, Ran Duan, Michael Elkin, Mohsen Ghaffari, David Harris, Zengfeng Huang, Yi Li, Nancy Lynch, Danupon Nanongkai, Yi Ouyang, Mira Radeva, Johannes Schneider, Roger Wattenhofer, and Xiaodi Wu. Special thanks to Cheng-Chiang Tsai who introduced the beauty of mathematics to me while I was in high school.

Finally, I would like to thank my family for their unconditional support. I cannot thank my grandma more for her devotion to us. Her homemade food is the best in the world.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Appendices

Abstract

Traditional studies of algorithms consider the sequential setting, where the whole input data is fed into a single device that computes the solution. Today, the network, such as the Internet, contains of a vast amount of information. The overhead of aggregating all the information into a single device is too expensive, so a distributed approach to solve the problem is often preferable. In this thesis, we aim to develop efficient algorithms for the following fundamental graph problems that arise in networks, in both sequential and distributed settings.

Graph coloring is a basic symmetry breaking problem in distributed computing. Each node is to be assigned a color such that adjacent nodes are assigned different colors. Both the efficiency and the quality of coloring are important measures of an algorithm. One of our main contributions is providing tools for obtaining colorings of good quality whose existence are non-trivial. We also consider other optimization problems in the distributed setting. For example, we investigate efficient methods for identifying the connectivity as well as the bottleneck edges in a distributed network. Our approximation algorithm is almost-tight in the sense that the running time matches the known lower bound up to a poly-logarithmic factor. For another example, we model how the task allocation can be done in ant colonies, when the ants may have different capabilities in doing different tasks.

The matching problems are one of the classic combinatorial optimization problems. We study the weighted matching problems in the sequential setting. We give a new scaling algorithm for finding the maximum weight perfect matching in general graphs, which improves the long-standing Gabow-Tarjan's algorithm (1991) and matches the running time of the best weighted bipartite perfect matching algorithm (Gabow and Tarjan, 1989). Furthermore, for the maximum weight matching problem in bipartite graphs, we give a faster scaling algorithm whose running time is faster than Gabow and Tarjan's weighted bipartite *perfect* matching algorithm.

Chapter 1

Introduction

Large networks arise in various scenarios such as the Internet, the cellular network, the social network, and the biological system. Traditional studies focus on developing efficient algorithms for problems arising in networks, when the whole graph data is fed into a single computational device. However, due to limited communication, the autonomy of the nodes, and the massive size of the network, a distributed model of computation is sometimes more realisitic. Instead of having a centralized coordinator, goals are achieved through coordination between the nodes. Nodes can communicate directly if there are links between them.

We study several basic graph problems in both sequential and distributed settings. The first part is dedicated to distributed graph coloring problems. The second part considers other distributed optimization problems. The third part studies the matching problems in the seqeuntial setting.

1. **Graph Coloring.** Consider that in a network, two adjacent nodes cannot broadcast at the same time. Protocols such as time division multiple access (TDMA) resolve this problem by assigning adjacent nodes with different time slots. Assigning the time slots for the nodes is equivalent to the graph coloring problem. Two measurements are considered. One is the efficiency, that is, to color the nodes using as few communication rounds as possible; another is the quality of coloring, that is, to use as few colors as possible. We study a variety of coloring problems in the distributed setting under both measurements.

2. **Other Distributed Optimization Problems.** When considering the optimization problems in the distributed setting, a simple algorithm is often needed under limited computation and communication. We study optimization problems in the distributed setting, including how to efficiently compute a minimum cut in a distributed setting and task allocation problems in ant colonies.

3. **Matchings.** The matching problems are classic problems in combinatorial optimization. They have various applications arising in task assignment, radar tracking systems, scheduling of a communication switch etc. Faster matching algorithms can lead directly to faster algorithms for solving many combinatorial optimization problems, such as Christofides' 3/2-approximate Metric TSP algorithm [22], the Chinese Postman problem [115], and undirected single-source shortest paths [58]. We study how fast we can compute weighted matchings in the sequential setting.

1.1 The Distributed Model

In the distributed setting, the underlying network is a graph $G = (V, E)$, where each node hosts a processor. The computation proceeds in synchronized rounds. In particular, at the beginning of each round, each node receives messages from its neighbors sent from the last round. Then, the nodes start to do computation. When the computation is finished, the node sends every neighbor a message. The messages sent to different neighbors may be different. The round ends when every node has finished sending the messages. The messages will reach their destinations in the beginning next round.

The time complexity is measured by the number of rounds. In the LOCAL model, we assume the message size is unbounded, whereas in the CONGEST model, each message is bounded by $O(\log n)$ bits [144]. We denote the number of vertices in G by n, the number of edges by m, the maximum degree by Δ, the diameter by D. *With high probability* (w.h.p.) means with probability $1 - 1/n^c$, for a fixed constant c. In most cases, we also assume that there is an $O(\log n)$-bits unique identifier, $\mathrm{ID}(u)$, associated with each vertex $u \in G$. In the problems we study, the input graph is the underlying network itself.

1.1.1 A Sample of Problems in Distributed Computing

An *independent set* is a set vertices where each pair is non-adjacent. A *maximal independent set* (MIS) is an independent set such that each vertex in G is in the set or is adjacent to at least one vertex in the set. The MIS problem is a natural problem that arises in distributed networks. For example, it arises when the nodes are to cluster themselves such that the cluster-heads are non-adjacent. Recent studies also pointed out the similarity between the MIS problem and the development of the fly's nervous system, when sensory organ precursor (SOP) cells are chosen [1]. Finding an MIS is a building block for many distributed algorithms [144] (e.g., the Moser-Tardos algorithm in Section 2).

A straightforward way to find the MIS is the following. Each active node sends its identifier to its neighbor. If a node has the smallest identifier among its active neighbors, then it selects itself to be in the MIS and then informs its neighbors. If a node or its neighbor has been selected to be in the MIS, then it becomes deactivated. If this step is repeated, all vertices will become deactivated eventually. The description of the algorithm can be implemented easily in the CONGEST model. However, such an algorithm may take time proportional to n, the number of nodes.

With the help of randomization, Luby [120] showed that an MIS can be computed in $O(\log n)$ rounds w.h.p. Such a time complexity has the meaning that the algorithm is *local* in the sense that each node does not have to know the information of the whole graph to find an MIS. One of the variants of Luby's algorithm is the following: Instead of using a fixed identifier in the above algorithm, each node generates an identifier randomly in each round. It selects itself to be in the MIS if it has the smallest identifier among its active neighbors. Luby's analysis shows such a modification would reduce the compleixty to $O(\log n)$.

Interestingly, at the time this thesis is written, there are no algorithms that have an faster asymptotic running time (in n) than Luby's algorithm, although there are faster algorithms when the degree, Δ, is bounded or when the graph contains some structure [7, 8, 10, 11, 156]. On the other hand, Kuhn, Moscibroda, and Wattenhofer [110] gave a $\Omega(\sqrt{\log n})$ lower bound for the MIS problem. Their lower bound holds under the LOCAL model and is immune to randomization.

Two problems are closely related to the MIS problem: the *maximal matching* (MM) and the

coloring problems. A *matching* is a set of vertex-disjoint edges. A *maximal matching* (MM) is a matching such that each edge in G is incident to some edge in the matching. An MIS algorithm can be used to find the MM by simulating it on the line graph of G. The coloring problems are to color the vertices (or edges) such that adjacent vertices (or edges) are not assgined the same color. In Part I of the thesis we will discuss more about these problems.

The MIS and the related problems can be computed without learning the whole graph. However, problems such as the minimum spanning tree (MST) are known to be *global*. Consider a ring with n nodes. Whether an edge uv is in the MST depends on whether the weight of uv is smallest among all the edges. Obviously, this information cannot be learned at u in less than $n/2$ rounds, even in the LOCAL model.

For the problems that require global knowledge, it would require $\Theta(D)$ rounds for each node to learn the whole graph. In $\Theta(D)$ rounds, problems in the LOCAL model become trivial, since after learning the whole information of the graph, the solution can be computed at a single node locally and broadcast back to the entire graph. Therefore, the interesting questions are how efficient the problems can be solved in the CONGEST model, when there are limits on the bandwidth. For the MST problem, Kutten and Peleg [114] gave an algorithm that runs in $O(\sqrt{n}\log^* n + D)$ rounds in the CONGEST model. The former term reflects the congestion bottleneck on the information needed. For the lower bounds, by reductions from communication complexity results, Das Sarma et al. [31] showed $\tilde{\Omega}(\sqrt{n} + D)$ lower bounds for a series of global problems, including MST, shortest path, minimum cuts, etc. In Chapter 6, we will give an almost-tight approximation algorithm for the minimum cut that runs in $\tilde{O}(\sqrt{n} + D)$ rounds.

1.1.2 Definitions for the Coloring Problems

In this section, we define several notations for the graph coloring problems. In the vertex coloring problem, we are given a graph $G = (V, E)$, where each vertex is associated with a palette $P(u)$. A coloring is proper if each vertex u is assigned with a color in $P(u)$ such that no adjacent vertices are assigned the same color. In the k-coloring problem, every vertex u can assign itself a color from palette $P(u) = \{1, 2, \ldots k\}$. The chromatic number $\chi(G)$ is defined to be the minimum k such that a k-coloring exists. An instance of the k-list-coloring

problem is one where each u has a palette of size k. A proper coloring in such an instance is called a k-list-coloring. Given G, if every instance of the k-list-coloring problem with respect to G has a proper coloring, then G is k-list-colorable. The list chromatic number $\chi_l(G)$ is defined to be minimum k such that G is k-list-colorable.

The notations for edge coloring are defined analogously. In the edge coloring problems, each edge e is associated with a palette $P(e)$. A coloring is proper if each edge e is assigned with a color in $P(e)$ such that no adjacent edges are assigned the same color. In k-edge-coloring problem, every edge e can assign itself a color from $P(e) = \{1, 2, \ldots k\}$. The chromatic index $\chi'(G)$ is defined to be the minimum k such that a proper k-edge-coloring exists. An instance of the k-edge-list-coloring problem is one where each edge e has a palette of size k. A proper coloring in such an instance is called a k-list-edge-coloring. Given G, if every instance of the k-list-coloring problem has a proper coloring, then G is k-list-edge-colorable. The list chromatic index $\chi'_l(G)$ is defined to be minimum k such that G is k-list-edge-colorable.

1.2 Overview of the Results

1.2.1 Graph Coloring

Graph coloring is an important symmetry-breaking primitive in distributed computing. The studies in distributed graph coloring can be divided into two lines, the deterministic algorithms and the randomized algorithms. The deterministic distributed coloring algorithm can be traced back to Cole and Vishkin [24], who devised an $O(\log^* n)$ algorithm for 3-coloring a cycle. Linial [118] showed that $O(1)$-coloring a cycle requires at least $\frac{1}{2}\log^* n - O(1)$ rounds for any function $f(\cdot)$. Goldberg and Plotkin [69] generalized the algorithm of [24] to the $(\Delta + 1)$-coloring problem in general graphs with a running time of $\Delta^{O(\Delta)} + O(\log^* n)$. Since then, a sequence of improvements [70, 111, 118, 140, 161] lead to an algorithm [10] that runs in $O(\Delta + \log^* n)$ time for the $(\Delta + 1)$-coloring problem. Panconesi and Srinivasan's determinsitic network decomposition approach [141] gives a $2^{O(\sqrt{\log n})}$ rounds algorithm for $(\Delta + 1)$-coloring.

For randomized algorithms, the seminal work of Luby [120] showed that the $(\Delta+1)$-coloring problem can be reduced to the MIS problem, which can be computed in $O(\log n)$ rounds.

Johansson [90] gives a $(\Delta+1)$-coloring algorithm that has the same time bound but with a smaller message compleixity than the MIS reduction. Schneider and Wattenhofer [157] showed that $(\Delta+1)$-coloring can be obtained in $O(\log\Delta+\sqrt{\log n})$ rounds. Barenboim et al. [11] gave an algorithm for $(\Delta+1)$-coloring running in $O(\log\Delta)+2^{O(\sqrt{\log\log n})}$ rounds using a graph-shattering technique.

Notice that for any graph G we know that $\chi(G)\leq\Delta+1$, as a $(\Delta+1)$-coloring can be obtained greedily. The bound is tight in general, since a $(\Delta+1)$-clique cannot be colored with Δ colors. If we are to trade the quality of coloring for the efficiency, then there are more efficient algorithms. For deterministic algorithms, Linial [118] and Szegedy and Vishwanathan [161] gave algorithms that run in $O(\log^* n)$ rounds for obtaining a $O(\Delta^2)$-coloring. Barenboim and Elkin [8] showed that an $O(\Delta^{1+\epsilon})$-coloring can be obtained determinsitically in polylogarithmic rounds, $O(\log\Delta\cdot\log n)$ rounds. For randomized algorithms, Kothapalli et al. [108] showed that an $O(\Delta)$-coloring can be obtained in $O(\sqrt{\log n})$ rounds. Schneider and Wattenhofer [157] showed that an $O(\Delta\log^{(k)} n+\log^{1+1/k} n)$-coloring can be obtained in $O(k)$ rounds. Barenboim et al. [11] showed the rounds needed to obtain such a coloring can be reduced to $2^{O(\sqrt{\log\log n})}$.

On the other hand, less was known from the distributed algorithm aspect when coloring graphs with $\chi(G)<\Delta+1$. Traditional studies on symmetry-breaking consider those problems whose solutions can be obtained via greedy methods, such as the Maximal Matching (MM) problem, the Maximal Independent Set (MIS) problem, and the $(\Delta+1)$-coloring problem. In these problems, given any partial solution to the problem, we can extend it to a complete solution without making regrets. Coloring a graph with girth at least 5 is an example which does not fall into this case. It was shown by Kim [104] that such graphs have $\chi(G)=(1+o(1))\Delta/\log\Delta$. It is clear that such a coloring is not greedily obtainable. Grable and Panconesi [73] gave a randomized algorithm for obtaining an $O(\Delta/\log\Delta)$-coloring in $O(\log n)$ rounds, provided $\Delta=(\log n)^{1+\Omega(1)}$. Another example is the $(\Delta+1)$-edge-coloring problem. It was shown by Vizing [170] that for any simple graph G, either $\chi'(G)=\Delta$ or $\chi'(G)=\Delta+1$. However, a $(\Delta+1)$-edge-coloring is not greedily obtainable. In fact, currently there are no known efficient distributed algorithms for obtaining such a coloring. A good appoximation of $(1+\epsilon)\Delta$-edge-coloring can be obtained in $O(\log n)$ rounds by using Dubhashi, Grable, and Panconesi's algorithm [39], provided that $\Delta=(\log n)^{1+\Omega(1)}$.

One of our contributions is to find general techniques for solving coloring problems without greedy algorithms, including the frugal coloring problem, the defective coloring problem, the list coloring problem, and the two problems mentioned above. The probabilistic method is often used to show the existence of the solution. However, the probabilistic arguments often rely on the Lovász Local Lemma (LLL), which is non-constructive. In Chapter 2, we propose tools for converting non-constructive proofs that use the LLL, to distributed algorithms for constructing the solutions efficiently.

The Lovász Local Lemma is often used to prove the existence of certain objects. Suppose there are bad events $A_1, \ldots, A_n$ that depend on some random variables. The goal is to show there exists an assignment to the variables that avoids all the bad events. The symmetric version of LLL tells us that if the probability of each event is bounded by $1/(e(d+1))$, where d is the maximum degree of the dependency graph of the events, then with positive probability all bad events can be avoided. We study a variety of coloring problems whose solutions are obtained by LLL. In these problems, it is often that each node is associated with a bad event, where each bad event is determined by the random variables in the neighborhood of its associated node. The probability of the bad events often have the form $\exp(-\Delta^{\Theta(1)})$ as a result of Chernoff-type concentration inequalities. One example is when tossing coins at each node v, we define the bad event A_v to be that the difference between the number of neighbors with a head and the number of neighbors with a tail exceeds $\Delta/4$. Therefore, $\Pr(A_v) = \exp(-O(\Delta))$. The union bound cannot be used to show all bad events can be avoided when Δ is sub-logarithmic in n. Since each bad event depends on the random variables in its neighborhood, $d = \text{poly}(\Delta)$, one can resort to the LLL. We develop tools for converting non-constructive proofs that use LLL, to distributed algorithms for constructing the solutions efficiently. Our distributed algorithm for the LLL circumvent the MIS computations in Moser and Tardos' algorithm [129]. This leads to a simpler and faster construction.

Chapter 3 and Chapter 4 are dedicated to efficient procedures for triangle-free graph coloring and edge coloring. The main technique used is the Rödl Nibble method, which is an iterative random process that partially colors the graph over the iterations. We show that $\chi(G) \leq (4 + o(1))\Delta/\log\Delta$ and $\chi_l(G) \leq (4 + o(1))\Delta/\log\Delta$ in triangle-free graphs. Previously, the best known bound is $(160 + o(1))\Delta/\log\Delta$ [125]. For the edge coloring problems, we study the $(1 + o(1))\Delta$-edge-coloring problem and $(1 + o(1))\Delta$-edge-list-coloring problem.

For triangle-free graphs, our iterative process takes $O(k + \log^* n)$ rounds to obtain a (Δ/k)-(list-)coloring for $k \leq \log \Delta/(4+\epsilon)$. For edge coloring problems, it takes $O(\log^* n)$ iterations to obtain an $(1+\epsilon)\Delta$-(list-)edge-coloring. For $\Delta \gg \log n$, we show each iteration succeeds with high probability and therefore the number of rounds equal to the number of iterations. For small values of Δ, we will have to resort to the Lovász Local Lemma. By applying our distributed algorithm for the LLL, we show the colorings can be obtained in $O(\log n)$ rounds.

In Chapter 5, we tackle one of the most basic problems in graph coloring, the $(\Delta+1)$-coloring problem. We show that in $(1-\epsilon)$-locally-sparse graphs and graphs whose arboricities are bounded by $(1-\epsilon)\Delta/2$, a $(\Delta+1)$-coloring can be obtained in $O(\log(1/\epsilon)) + e^{O(\sqrt{\log\log n})}$ rounds. This result also shows a separation between the $(2\Delta-1)$-edge-coloring problem and the MM problem. The former can be solved in $e^{O(\sqrt{\log\log n})}$ rounds because its line graph is roughly $\frac{1}{2}$-locally-sparse, while the latter is known to have a lower bound of $\Omega(\sqrt{\log n})$ [110].

1.2.2 Other Distributed Optimization Problems

The Minimum Cut Problem The minimum cut problem is a fundamental problem in graph algorithms and network design. It determines, e.g., the network vulnerability and the limits to the speed at which information can be transmitted. Given a weighted undirected graph $G = (V, E)$, a cut $C = (S, V \setminus S)$ where $\emptyset \subset S \subset V$, is a partition of vertices into two non-empty sets. The weight of a cut, $w(C)$, is defined to be the sum of the edge weights crossing C. The minimum cut problem is to find a cut with the minimum weight, λ. The exact version of the problem as well as the approximate version have been studied for many years [56, 94, 96, 97, 99, 121, 132, 160] in the context of centralized models of computation, resulting in nearly linear time algorithms [96, 97, 102, 121].

Elkin [46] and Das Sarma et al. [31] addressed the problem in the distributed, synchronous message-passing model. The problem has trivial time complexity of $\Theta(D)$ (unweighted diameter) in the LOCAL model, where the message size is unlimited.

In the CONGEST model, Prichard and Thurimella's 2-edge-connected and 3-edge-connected components algorithm [151] can be used to find the exact minimum cut for $\lambda \leq 3$. Ghaffari and Kuhn [65] gave an algorithm that finds a cut of size at most $O(\epsilon^{-1}\lambda)$ with high probability in $O(D) + O(n^{1/2+\epsilon} \log^3 n \log\log n \log^* n)$ time. They also gave an algorithm that finds a cut

of size at most $(2+\epsilon)\lambda$ w.h.p. in $O((D+\sqrt{n}\log^* n)\log^2 n \log\log n \frac{1}{\epsilon^5})$ time. Das Sarma et al. [31] showed α-approximating the minimum cut requires $\tilde{\Omega}(D+\sqrt{n})$ rounds for any $\alpha \geq 1$.

In Chapter 6, we present a distributed algorithm for finding an $(1+\epsilon)$-approximate minimum cut in $O((D+\sqrt{n}\log^* n)\epsilon^{-5}\log^3 n)$ rounds with high probability. Our algorithm draws a connection between Thorup's tree packing lemma and Matula's contraction algorithm. It can be also be implemented easily in the sequential setting in $O(m+n\epsilon^{-7}\log^3 n)$ time, which may be simpler than Karger's $O(m+n\epsilon^{-4}\log^3 n)$-time approximation algorithm [96], which uses Gabow's exact minimum cut algorithm as a subroutine [56]. Moreover, we show that an exact minimum cut can be computed in $O((\sqrt{n}\log^* n+D)\lambda^4\log^2 n)$ rounds in the distributed setting.

Task Allocation Problem Many biological systems are similar to the model of distributed computation in the sense that they operate without a centralized coordinator [136]. We consider task allocation problems in ant colonies. Cornejo et al. [26] first considered modeling the task allocation problem from a distributed computing perspective. In their model, each task i is associated with a fixed demand d_i. Each ant provides a unit supply of energy to the task. Also, the ants are able to sense from each task whether it is undersatified or oversatisfied. They showed that ants can solve the task allocation problem using constant number of states and $O(|T|)$ bits of memory in $O(|T|\log|A|)$ rounds, where T is the set of tasks and A is the set of ants.

When there are individual variations among the ants, where the energies provided by the ants to the tasks may be different, [26] pointed out that finding a feasible allocation to the tasks is NP-hard by a reduction from the partition problem. In Chapter 7, we give a very simple mechanism for ants to converge to a solution that approximately satisfies the demands. In particular, for any $\epsilon \geq 0$, we show that after $O(|A|^{1-\epsilon})$ rounds, the ants converge to a solution with an $O(W|A|^{1/2+\epsilon})$ additive-error with probability $1-O(1/|A|^{\epsilon})$, where W is ratio of the largest to the smallest energies provided. We also show a better bound for the case when there are no individual variations and when the ants behave determinsitically. The ants converges to an optimal solution in $O(\epsilon^{-2}|T|\log|T|)$ rounds with an additive error of $\epsilon d_{\max}+o(1)$, where $d_{\max}$ is the largest demand of the tasks.

The techniques involved for both problems are related to the multiplicative weight update

method. For the minimum cut problem, the greedy tree packing step is implicitly doing the multiplicative weight update method for finding a near-optimal tree packing. For the task allocation problem, we also transform the idea of the multiplicative weight update method into a simple strategy with a modification where the weights are updated stochastically.

1.2.3 Matchings

We consider the matching problems in the sequential setting. Given a weighted graph, the maximum weight perfect matching (MWPM) problem is to find a perfect matching with the maximum weight, while the maximum weight matching (MWM) problem is to find a (non-necessarily perfect) matching with the maximum weight. The MWM problem and the MWPM are known to be reducible to each other. Given an algorithm for the MWM problem running in $f(n, m, N)$ time, where N is the largest weight of the edges, it can be used to solve the MWPM problem in $f(n, m, O(nN))$ time. On the other hand, given an algorithm for the MWPM problem running in $g(n, m, N)$ time, it can be used to solve the MWM problem in $g(O(n), O(m), N)$ time.

Gabow amd Tarjan [61] gave a scaling algorithm for the MWPM problem running in $O(m\sqrt{n\alpha(n)\log n}\log(nN))$ time. For bipartite cases, Gabow and Tarjan [60] gave a scaling algorithm for MWPM problem running in $O(m\sqrt{n}\log(nN))$ time.

In Chapter 8, we gave a scaling algorithm for MWPM running in $O(m\sqrt{n}\log(nN))$ time for general graphs, which matches best known bound for bipartite graphs. In Chapter 9, we gave a scaling algorithm for finding the MWM in bipartite graphs. Our algorithm runs in $O(m\sqrt{n}\log N)$ time, which is asymptotically faster than applying Gabow and Tarjan's MWPM algorithm to find the MWM, when $\log N = o(\log(nN))$.

Part I

Coloring

Chapter 2

Distributed Algorithms for the Lovász Local Lemma

2.1 Introduction

Consider a system $\mathcal{P}$ of independent random variables and a set $\mathcal{A}$ of n *bad* events, where each $A \in \mathcal{A}$ depends solely on some subset $\text{vbl}(A) \subseteq \mathcal{P}$. For example, in a hypergraph 2-coloring instance, $\mathcal{P}$ represents the vertex colors and $\mathcal{A}$ the events in which an edge is monochromatic. The dependency graph $G_{\mathcal{A}} = (\mathcal{A}, \{(A, B) \mid \text{vbl}(A) \cap \text{vbl}(B) \neq \emptyset\})$ includes edges between events if and only if they depend on at least one common variable. Let $\Gamma(A)$ be A's neighborhood in $G_{\mathcal{A}}$ and $\Gamma^+(A) = \Gamma(A) \cup \{A\}$ be its inclusive neighborhood. The (general, asymmetric) LLL states [48,158] that if there is a function $x : \mathcal{A} \to (0, 1)$ such that

$$\Pr(A) \leq x(A) \cdot \prod_{B \in \Gamma(A)} (1 - x(B))$$

then $\Pr(\bigcap_{A \in \mathcal{A}} \overline{A}) > 0$, that is, there is a *satisfying assignment* to the underlying variables in which no bad events occur. The symmetric LLL is a useful corollary of the general

LLL. If p and d are such that $\Pr(A) \leq p$ and $|\Gamma(A)| \leq d$ for all A, and $ep(d+1) < 1$, then $\Pr(\bigcap_{A \in \mathcal{A}} \overline{A}) > 0$. For example, consider a hypergraph in which each edge contains k vertices and intersects at most $d < 2^{k-1}/e - 1$ other edges. Under a uniformly random color assignment $\mathcal{P} \to \{\text{red, blue}\}$ the probability an edge is monochromatic is $p = 2^{-(k-1)}$, so $ep(d+1) < 1$. The symmetric LLL proves the *existence* of a satisfying color assignment but does not yield an efficient algorithm to find one. Beginning with Alon [2] and Beck [12], a long line of research has sought to find efficient (and ideally deterministic) algorithms for computing satisfying assignments [2,12,19,29,75–78,106,124,127–129,143,159]. Most of these results required a major weakening of the standard symmetric LLL constraint $ep(d+1) < 1$. In many applications we consider, the bad events are that the sum of $d^{\Theta(1)}$ random variables deviates away from its expectation. So the probability they are violated is often bounded by Chernoff-type tail bounds, e.g. $\exp(-d^{\Theta(1)})$.

In a relatively recent breakthrough, Moser and Tardos [129] gave an *algorithmic* proof of the general asymmetric LLL, with no weakening of the parameters. Their algorithm is simple though the analysis is not trivial. At initialization the algorithm chooses a random assignment to the variables $\mathcal{P}$. Call an event $A \in \mathcal{A}$ *violated* if it occurs under the current assignment to the variables. Let $\mathcal{F} \subseteq \mathcal{A}$ be the set of violated events. The algorithm repeatedly chooses some $A \in \mathcal{F}$ and *resamples* the variables in $\text{vbl}(A)$, until $\mathcal{F} = \emptyset$.

The Distributed LLL Problem We consider Linial's LOCAL model [144] of distributed computation in which the distributed network is identical to the dependency graph. In other words, each node $A \in \mathcal{A}$ hosts a processor, which is aware of n, the degree bound d, and its neighborhood $\Gamma(A)$. Computation proceeds in synchronized rounds in which each node may send an unbounded message to its neighbors. *Time* is measured by the number of rounds; computation local to each node is free. Upon termination each node A must commit to an assignment to its variables $\text{vbl}(A)$ that is consistent with its neighbors, i.e., the nodes must collectively agree on a satisfying assignment to $\mathcal{P}$ avoiding all bad events. We consider the LOCAL model because we will need to send the assignment of $\text{vbl}(A)$ in one message.

Moser and Tardos proposed a parallel version of their resampling algorithm (Algorithm 1), which can easily be implemented in the LOCAL model. Let $G_{\mathcal{F}}$ be the graph induced by the violated events $\mathcal{F}$ under the current variable assignment. They proved that $O(\log_{1/ep(d+1)} n)$

iterations of Algorithm 1 suffice to avoid all bad events with probability $1 - 1/\operatorname{poly}(n)$, i.e., $O(\log n)$ iterations suffice if $ep(d+1)$ is bounded away from 1[1]. (For the sake of a simpler presentation we shall state many results in the symmetric LLL language. Our algorithms and Moser-Tardos work for the asymmetric LLL as well.) Moser and Tardos suggested using Luby's randomized MIS algorithm [120], which runs in $\Theta(\log n)$ rounds w.h.p. (which can also be achieved by [3]), for a total running time of $\Theta(\log n \cdot \log_{1/ep(d+1)} n)$. This is, intuitively, a very wasteful LLL algorithm since nodes spend nearly all their time computing MISs rather than performing resampling steps. For certain values of d the running time can be improved by plugging in an MIS algorithm running in $O(d + \log^* n)$ time [10] or $O(\log^2 d) + \exp(O(\sqrt{\log \log n}))$ time w.h.p. [11].[2] However, it is not possible to find an MIS in constant time. Kuhn et al. [109, 110] gave an $\Omega(\min\{\log d, \sqrt{\log n}\})$ lower bound on the complexity of MIS and other symmetry-breaking problems.

Initialize a random assignment to the variables $\mathcal{P}$.
while $\mathcal{F} \neq \emptyset$ **do**
  Compute a maximal independent set $\mathcal{I}$ in $G_{\mathcal{F}}$.
  Resample each variable in $\operatorname{vbl}(\mathcal{I}) = \bigcup_{A \in \mathcal{I}} \operatorname{vbl}(A)$.
end while

Algorithm 1: The Moser-Tardos Parallel Resampling Algorithm. Here $\mathcal{F}$ is the set of bad events occurring under the current variable assignment and $G_{\mathcal{F}}$ is the dependency graph induced by $\mathcal{F}$.

New Results We give a new distributed LLL algorithm in the Moser-Tardos resampling framework that avoids the computation of MISs altogether. Due to its simplicity we are happy to display the algorithm in its entirety. We assume that nodes possess unique IDs, which could be assigned in an adversarial manner. Let $\Gamma_{\mathcal{F}}(A)$ be A's neighborhood in $G_{\mathcal{F}}$.

One can see that $\mathcal{I}$ is computed in one round: each node A tells its neighbors whether $A \in \mathcal{F}$ under the current variable assignment. Once A receives messages from all neighbors it can determine if $\operatorname{ID}(A)$ is a local minimum in $G_{\mathcal{F}}$. We prove that under the slightly stronger criterion $epd^2 < 1$, this algorithm halts in $O(\log_{1/epd^2} n)$ steps w.h.p. Most applications of the LLL satisfy the $epd^2 < 1$ criterion, though not all. We give another distributed

[1]Note that $\log_{1/ep(d+1)} n$ could be sublogarithmic or superlogarithmic depending on how close $ep(d+1)$ is to 0 or 1.

[2]These MIS algorithms are significantly more complex than Luby's and use larger messages.

Initialize a random assignment to the variables $\mathcal{P}$

while $\mathcal{F} \neq \emptyset$ **do**
  Let $\mathcal{I} = \{A \in \mathcal{F} \mid \mathrm{ID}(A) = \min\{\mathrm{ID}(B) \mid B \in \Gamma^+_{\mathcal{F}}(A)\}\}$
  Resample $\mathrm{vbl}(\mathcal{I}) = \bigcup_{A \in \mathcal{I}} \mathrm{vbl}(A)$.
end while

Algorithm 2: A Simple Distributed LLL Algorithm

LLL algorithm in the resampling framework that finds a satisfying assignment in $O(\log^2 d \cdot \log_{1/ep(d+1)} n)$ time under the usual $ep(d+1) < 1$ criterion.

We show that faster algorithms exist when the condition $ep(d+1) < 1$ is replaced by a stronger condition $p \cdot f(d) < 1$, where $f(d)$ is a faster growing function than $e(d+1)$. However, it is not clear whether there exists $f(d)$ so that the LLL can be solved in sublogarithmic time in n, independent of d. Moser and Tardos observed that any parallel algorithm in the resampling framework requires $\Omega(\log_{1/p} n)$ resampling steps, even if the dependency graph has no edges. We combine the resampling framework with a locality approach to give an $O(\log n / \log\log n)$ algorithm for an exponential function $f(d)$. On the other hand, we prove that no constant time distributed LLL algorithm exists and that the LLL for any $f(d)$ requires $\Omega(\log^* n)$ time.

New Applications Existential results in graph coloring [125] (those taking the *Rödl nibble* approach) can often be phrased as distributed algorithms in which each step succeeds with some tiny but non-zero probability, as guaranteed by the LLL. By using our distributed LLL algorithms we are able to solve a number of graph coloring problems in $O(\log n)$ time or faster.[3] Some of these applications require minor changes to existing algorithms while others are quite involved. Below Δ is the maximum degree, and $\epsilon > 0$ an arbitrarily small parameter.

Frugal Coloring A k-frugal vertex coloring is one in which each color appears at most k times in the neighborhood of any vertex. Pemmaraju and Srinivasan [145] showed

[3]Suppose H is both the distributed network and the graph to be colored. When invoking the LLL, the dependency graph $G_{\mathcal{A}}$ is not *identical* to H. Typically bad events in $\mathcal{A}$ are associated with H-vertices and two bad events are adjacent in $G_{\mathcal{A}}$ only if the corresponding vertices are at distance $O(1)$ in H. Thus, a distributed LLL algorithm for $G_{\mathcal{A}}$ can be simulated in H with an $O(1)$ slowdown.

the existence of $(\Delta+1)$-colorings that are $O(\log^2\Delta/\log\log\Delta)$-frugal, and proved that $(\log\Delta\cdot\log n/\log\log n)$-frugal colorings could be computed in $O(\log n)$ time. With some modifications to their proof we show that a $O(\log^2\Delta/\log\log\Delta)$-frugal $(\Delta+1)$-coloring can be computed in $O(\log n)$ time. Notice that the best existential bound on the frugality for $(\Delta+1)$-coloring is $O(\log\Delta/\log\log\Delta)$ by Molloy and Reed [126].

Hind, Molloy, and Reed [81] showed there exist β-frugal, $O(\Delta^{1+\frac{1}{\beta}})$-colorings by using the asymmetric LLL. We show how to turn their proof into a distributed algorithm that runs in $O(\log n\cdot\log^2\Delta)$ time.

Girth 4 and 5 Kim [104] showed that there exists an $(1+\epsilon)\Delta/\ln\Delta$-coloring for graphs of girth 5. In Chapter 3, we will show that triangle-free graphs have $(4+\epsilon)\Delta/\ln\Delta$-colorings. Also, we gave distributed algorithms that run in $O(\log n)$ rounds for both problems using our LLL algorithms.

Edge Coloring Dubhashi et al. [39] gave a $(1+\epsilon)\Delta$-edge-coloring algorithm running in $O(\log n)$ time, provided that $\Delta=(\log n)^{1+\Omega(1)}$ is sufficiently large relative to n. In Chapter 4, we apply our LLL algorithm to show that $(1+\epsilon)\Delta$-edge-coloring can be obtained in $O(\log^*\Delta+\log n/\Delta^{1-o(1)})$ rounds for $\Delta\geq\Delta_\epsilon$, where Δ_ϵ is a sufficiently large constant depending on ϵ.

List-Coloring Suppose each vertex is issued a list of $(1+\epsilon)D>D_\epsilon$ colors such that each color appears in at most D lists in the neighborhood of any vertex, where D_ϵ is a sufficiently large constant depending on ϵ. (D need not be close to the degree Δ.) Reed and Sudakov [153] proved that $(1+\epsilon)D$-list-colorings exist. We show how to construct them in $O(\log^* D+\log n/D^{1-o(1)})$ time. Furthermore, for *any* D and any constant $\epsilon>0$, we show that $(2e+\epsilon)D$-list-colorings can be obtained in $O(\log n)$ time.

Defective Coloring An f-defective coloring is one in which a vertex may share its color with up to f neighbors. Barenboim and Elkin [9], and implicitly, Kuhn and Wattenhofer [111] gave an $O(1)$ time procedure to compute a $O(\log n)$-defective $O(\Delta/\log n)$-coloring. We prove that for any $f>0$, an f-defective $O(\Delta/f)$-coloring can be computed in $O((\log n)/f)$ time.

2.2 Preliminaries

Let $\Gamma^r(A)$ be the r-neighborhood of A (the set of nodes at distance at most r from A, excluding A) and $\Gamma^{r+}(A) = \Gamma^r(A) \cup \{A\}$ be its inclusive r-neighborhood. A node set in the subscript indicates a restriction of the neighborhood to that set, e.g., $\Gamma^{2+}_{\mathcal{F}}(A) = \Gamma^{2+}(A) \cap \mathcal{F}$.

Consider an execution of a Moser-Tardos-type resampling algorithm. Let $C : \mathbb{N} \to \mathcal{A}$ be such that $C(i)$ is the ith event selected by the algorithm for resampling; C is called the *record* of the execution. (If the algorithm selects events in independent batches then the events in each batch can be listed arbitrarily.) A *witness tree* $\tau = (T, \sigma_T)$ is a finite rooted tree where $\sigma_T : V(T) \to \mathcal{A}$ labels each vertex in T with an event such that the children of $u \in T$ receive labels from $\Gamma^+(\sigma_T(u))$. A *2-witness tree* $\tau = (T, \sigma_T)$ is defined in the same way except that the children of $u \in T$ may receive labels from $\Gamma^{2+}(\sigma_T(u))$. A witness tree (or 2-witness tree) is *proper* if the children of a vertex receive distinct labels.

Given a record C, the witness tree $\tau_C(t)$ is constructed as follows. First, create a root node labelled $C(t)$. Looking backward in time, for each $i = t-1, t-2, \ldots, 1$, check if an existing node is labeled with an event from $\Gamma^+(C(i))$. If so, let u be one of the *deepest* such nodes. Create a new node v labeled $C(i)$ and make it a child of u. Given a witness tree τ, we say τ *occurs in* C if there exists an index t such that $\tau_C(t) = \tau$. Moser and Tardos proved the following lemma:

Lemma 2.2.1. *Let τ be a fixed witness tree and C be the record produced by the algorithm.*

1. If τ occurs in C, then τ is proper.
2. The probability that τ occurs in C is at most $\prod_{v \in V(\tau)} \Pr(\sigma_T(v))$.

Similarly, for $r \geq 2$, we can define an r-witness tree $\tau^r_C(t)$ in the same way except that in each step we attach a node labelled $C(i)$ to the deepest node among nodes labelled $\Gamma^{r+}(C(i))$. Also, we say τ *r-occurs in* C if there exists $t \in \mathbb{N}$ such that $\tau^r_C(t) = \tau$. Then Lemma 2.2.1 holds analogously:

Lemma 2.2.2. *Let τ be a fixed r-witness tree and C be the record produced by the algorithm.*

1. If τ r-occurs in C, then τ is proper.
2. The probability that τ r-occurs in C is at most $\prod_{v \in V(\tau)} \Pr(\sigma_T(v))$.

2.3 Algorithms

Recall that the parallel/distributed Moser-Tardos algorithm iteratively selects maximal independent sets (MIS) of violated events for resampling. They proved that if there is some slack in the general LLL preconditions then the algorithm terminates in $O(\log n)$ rounds of MIS.

Theorem 2.3.1. *(Moser and Tardos) Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If there exists an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that*

$$\forall A \in \mathcal{A} : \Pr(A) \leq (1 - \epsilon)x(A) \prod_{B \in \Gamma(A)} (1 - x(B)),$$

then the probability any bad event occurs after k resampling rounds of Algorithm 1 is at most $(1 - \epsilon)^k \sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

In other words, if $x(A)$ is bounded away from 1 then $O(\log_{\frac{1}{1-\epsilon}} n)$ resampling rounds suffice, w.h.p. A distributed implementation of this algorithm takes $O(\log_{\frac{1}{1-\epsilon}} n \cdot \mathrm{MIS}(n, d))$, where d is the maximum degree of $G_{\mathcal{A}}$ and $\mathrm{MIS}(n, d)$ is the time needed to find an MIS in an n-vertex degree-d graph. It is known that $\mathrm{MIS}(n, d) = \Omega(\min\{\sqrt{\log n}, \log d\})$ [109, 110]. Our algorithms avoid the computation of MISs. In Section 2.3.1 we analyze the simple distributed LLL algorithm presented in the introduction, which requires slightly weakening the general LLL conditions. In Section 2.3.2 we present an algorithm that works for the standard LLL conditions but is slower by a $O(\log^2 d)$ factor.

2.3.1 A Simple Distributed Algorithm

Recall that in each round of Algorithm 2, a violated event $A \in \mathcal{F}$ is selected for resampling if $\mathrm{ID}(A)$ is a local minimum in the violated subgraph $G_{\mathcal{F}}$. In order to analyze this algorithm in the witness tree framework we must establish some connection between the depth of witness trees and the number of rounds of resampling. Lemma 2.3.2 will let us make such a connection.

Lemma 2.3.2. *Suppose an event A is resampled in round $j > 1$ of Algorithm 2. There must exist some $B \in \Gamma^{2+}(A)$ resampled in round $j-1$.*

Proof. Let $\mathcal{F}'$ and $\mathcal{F}$ be the violated event sets just before and after the resampling step at round $j-1$. If A is not in $\mathcal{F}'$ but is in $\mathcal{F}$ then its variables $\text{vbl}(A)$ must have been changed in round $j-1$, which could only occur if some $B \in \Gamma(A)$ were resampled. Now suppose A is in both $\mathcal{F}'$ and $\mathcal{F}$. It was not resampled in round $j-1$ but was in round j, meaning $\text{ID}(A)$ is not a local minimum in $\Gamma_{\mathcal{F}'}(A)$ but is a local minimum in $\Gamma_{\mathcal{F}}(A)$. This implies that some neighbor $B \in \Gamma(A)$ with $\text{ID}(B) < \text{ID}(A)$ is in $\mathcal{F}'$ but not $\mathcal{F}$, which could only occur if some $C \in \Gamma^{+}(B) \subseteq \Gamma^{2+}(A)$ were resampled in round $j-1$. □

We can now proceed to bound the number of rounds of Algorithm 2 needed to find a satisfying assignment.

Theorem 2.3.3. *(Asymmetric LLL) Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If there exists an assignment of reals $x : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} : \Pr(A) \leq (1-\epsilon)x(A) \prod_{B \in \Gamma^2(A)} (1 - x(B)),$$

then the probability any bad event occurs after k resampling rounds of Algorithm 2 is at most $(1-\epsilon)^k \sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

Note the difference with Theorem 2.3.1 is that the product is over all $B \in \Gamma^2(A)$ not $B \in \Gamma(A)$.

Corollary 2.3.4. *(Symmetric LLL) Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables, such that for $\forall A \in \mathcal{A}$*

1. *$Pr(A) \leq p < 1$, and*
2. *A shares variables with at most d of the other events.*

If $epd^2 < 1$, then w.h.p. none of the bad events occur after $O(\log_{\frac{1}{epd^2}} n)$ rounds of Algorithm 2.

Proof. Setting $x(A) = 1/d^2$ and $\epsilon = 1 - epd^2$ in Theorem 2.3.3, we have

$$\begin{aligned}(1-\epsilon)x(A)\prod_{B\in\Gamma^2(A)}(1-x(B)) &\geq \frac{1-\epsilon}{d^2}\cdot\left(1-\frac{1}{d^2}\right)^{|\Gamma^2(A)|}\\ &\geq \frac{1-\epsilon}{d^2}\left(1-\frac{1}{d^2}\right)^{(d^2-1)} \geq \frac{1-\epsilon}{ed^2} \geq p \geq \Pr(A).\end{aligned}$$

Therefore, the probability a bad event occurs after k rounds of resampling is at most $(1-\epsilon)^k \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)} = (1-\epsilon)^k n/(d^2-1)$, which is $1/\operatorname{poly}(n)$ if $k = O(\log_{\frac{1}{1-\epsilon}} n) = O(\log_{\frac{1}{epd^2}} n)$. □

Following Moser and Tardos [129] we analyze the following Galton-Watson process for generating a r-witness tree T. Fix an event $A \in \mathcal{A}$. Begin by creating a root for T labelled A. To shorten the notation, we let $[v] := \sigma_T(v)$. In each subsequent step, consider each vertex v created in the previous step. For each $B \in \Gamma^{r+}([v])$, independently, attach a child labelled B with probability $x(B)$ or skip it with probability $1 - x(B)$. Continue the process until no new vertices are born. We prove a lemma analogous to one in [129].

Lemma 2.3.5. *Let τ be a fixed proper r-witness tree with its root vertex labelled A. The probability p_τ that the Galton-Watson process yields exactly the tree τ is*

$$p_\tau = \frac{1-x(A)}{x(A)}\prod_{v\in V(\tau)} x'([v])$$

where $x'(B) = x(B)\cdot\Pi_{C\in\Gamma^r(B)}(1-x(C))$.

Proof. Let $W_v \subseteq \Gamma^{r+}([v])$ denote the set of inclusive r-neighbors of $[v]$ that do not occur as

a label of some child node of v. Then,

$$\begin{aligned}
p_\tau &= \frac{1}{x(A)} \cdot \prod_{v \in V(\tau)} \left(x([v]) \cdot \prod_{u \in W_v} (1 - x([u]) \right) \\
&= \frac{1 - x(A)}{x(A)} \cdot \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \cdot \prod_{u \in \Gamma^{r+}([v])} (1 - x([u])) \right) \\
&= \frac{1 - x(A)}{x(A)} \cdot \prod_{v \in V(\tau)} \left(x([v]) \cdot \prod_{u \in \Gamma^{r}([v])} (1 - x([u])) \right) \\
&= \frac{1 - x(A)}{x(A)} \cdot \prod_{v \in V(\tau)} x'([v])
\end{aligned}$$

□

Lemma 2.3.6. *If for all $A \in \mathcal{A}$, we have $\Pr(A) \leq (1-\epsilon)x(A) \cdot \prod_{B \in \Gamma^r(A)}(1 - x(B))$, then the probability that any r-witness tree of size at least k occurs is at most $(1-\epsilon)^k \cdot \sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.*

Proof. Let $\mathcal{T}_A^r(k)$ denote the infinite set of r-witness trees having root labelled A and containing at least k vertices. By Lemma 2.2.2 and the union bound, the probability there exists a violated event after k resampling rounds is at most

$$\begin{aligned}
&\sum_{A \in \mathcal{A}} \sum_{\tau \in \mathcal{T}_A^r(k)} \Pr(\tau \text{ } r\text{-occurs in } C) \\
&\leq \sum_{A \in \mathcal{A}} \sum_{\tau \in \mathcal{T}_A^r(k)} \prod_{v \in V(\tau)} \Pr([v]) && \text{by Lemma 2.2.2} \\
&\leq \sum_{A \in \mathcal{A}} \sum_{\tau \in \mathcal{T}_A^r(k)} \prod_{v \in V(\tau)} (1 - \epsilon) x'([v]) && \text{cond. of Thm 2.3.3} \\
&\leq (1-\epsilon)^k \sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)} \sum_{\tau \in \mathcal{T}_A^r(k)} p_\tau && \text{by Lemma 2.3.5} \\
&\leq (1-\epsilon)^k \sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)}
\end{aligned}$$

The last inequality follows since the Galton-Watson process grows exactly one tree. □

Let C be the record of Algorithm 2 and S_j be the segment of the record corresponding to

resamplings in round j. The following lemma relates the number of resampling rounds with the occurence of 2-witness trees.

Lemma 2.3.7. *If there is still a violated event after k resampling rounds in Algorithm 2 then some 2-witness tree of size at least k occurs in C.*

Proof. Let A_k be any event in S_k and t be its position in the record C. By Lemma 2.3.2 there exist events $A_{k-1}, \ldots, A_1$ in $S_{k-1}, \cdots, S_1$ such that for all $j < k$, $A_j \in \Gamma^{2+}(A_{j+1})$. This implies that $A_{k-1}, \ldots, A_1$ are mapped to distinct nodes in the 2-witness tree $\tau_C(t)$, whose root is labeled A_k. □

Therefore, by Lemma 2.3.7, if there is a violated event after k resampling rounds, then a 2-witness tree of size at least k occurs. However, by Lemma 2.3.6, it happens with probability at most $(1-\epsilon)^k \cdot \sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$. Thus, Theorem 2.3.3 holds. Note that if $x(A)$ is bounded away from 1, then after $O(\log_{\frac{1}{1-\epsilon}} n)$ rounds, w.h.p. no bad event occurs.

2.3.2 Resampling by Weak MIS

In this section we analyze the efficiency of Moser and Tardos's Algorithm 1 when a new *weak MIS* procedure (Algorithm 3) is used in lieu of an actual MIS. The Weak-MIS procedure produces, in $O(\log^2 d)$ time, an independent set S such that the probability that a node is *not* in $\Gamma^+(S) = S \cup \Gamma(S)$ is $1/\operatorname{poly}(d)$. The procedure consists of $O(\log d)$ iterations where the probability that a vertex avoids $\Gamma^+(S)$ is constant per iteration. Each iteration consists of $\log d$ phases where, roughly speaking, the goal of phase i is to eliminate vertices with degree at least $d/2^i$ with constant probability. Each phase is essentially one step of Luby's MIS algorithm, though applied only to a judiciously chosen subset of the vertices. See Algoirthm 3.

Our main results are as follows.

Theorem 2.3.8. *(Assymmetric LLL) Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If there exists an assignment of reals $x : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} : \Pr(A) \leq (1-\epsilon)x(A) \prod_{B \in \Gamma(A)} (1 - x(B)),$$

then the probability any bad event occurs after k resampling rounds using the Weak-MIS algorithm is at most $n(\frac{1}{d+1})^k + (1-\epsilon)^{k/2}\sum_{A\in\mathcal{A}}\frac{x(A)}{1-x(A)}$.

Corollary 2.3.9. *(Symmetric LLL) Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables, such that for $\forall A \in \mathcal{A}$,*

1. *$Pr(A) \leq p < 1$, and*
2. *A shares variables with at most d of the other events.*

If $ep(d+1) < 1$, then w.h.p. none of the bad events occur after $O(\max(\log_{d+1} n, \log_{\frac{1}{ep(d+1)}} n))$ Weak-MIS resampling rounds.

Corollary 2.3.9 follows directly by plugging in $x(A) = 1/(d+1)$ for all $A \in \mathcal{A}$ and $k = O(\max(\log_{d+1} n, \log_{\frac{1}{ep(d+1)}} n))$. Notice that if $\frac{1}{ep(d+1)} > d+1$, we can apply the faster simple distributed algorithm, so the running time in Corollary 2.3.9 will be dominated by $O(\log_{\frac{1}{ep(d+1)}} n \cdot \log^2 d)$.

$S \leftarrow \emptyset$
for iteration $1 \ldots, t = 4e^2 \ln(2e(d+1)^4)$ **do**
  $G' \leftarrow G_{\mathcal{F}} \setminus \Gamma^+(S)$
  for phase $i = 1 \ldots \lceil \log d \rceil$ **do**
    $V_i \leftarrow \{v \in G' \mid \deg_{G'}(v) \geq d/2^i\}$.
    For each vertex $v \in G'$, set $b(v) \leftarrow \begin{cases} 1 & \text{with probability } p_i = 1/(\frac{d}{2^{i-1}}+1) \\ 0 & \text{otherwise} \end{cases}$
    For each vertex $v \in G'$, if $b(v) = 1$ and $b(w) = 0$ for all $w \in \Gamma_{G'}(v)$, set $S \leftarrow S \cup \{v\}$.
    $G' \leftarrow G' \setminus (\Gamma^+(S) \cup V_i)$ (i.e., remove both $\Gamma^+(S)$ and V_i from G'.)
  end for
  Let S' be the (isolated) vertices that remain in G'.
  Set $S \leftarrow S \cup S'$
end for
return S

Algorithm 3: Weak-MIS

Consider the first iteration of the Weak-MIS algorithm. For each phase i, G' is the subgraph of $G_{\mathcal{F}}$ containing vertices with degree at most $d/2^i$ and not adjacent to the independent set S. Let $V_i = \{v \in G' \mid \deg_{G'}(v) \geq d/2^i\}$. Note that every vertex in $G_{\mathcal{F}}$ must end up isolated in S' or one of the V_i's. Let (u, v) be an edge in G'. Following Peleg's analysis [144], define $\mathcal{E}(u, v)$ to be the event that at phase i, $b(u) = 0$ and $b(v) = 1$ and for all other neighbors x of u and v, $b(x) = 0$. Define $\mathcal{E}(u) = \bigcup_{v \in \Gamma_{G'}(u)} \mathcal{E}(u, v)$ to be the event that exactly one neighbor joins S in this phase. Since these events are disjoint, we have $\Pr(\mathcal{E}(u)) = \sum_{v \in \Gamma_{G'}(u)} \Pr(\mathcal{E}(u, v))$.

Lemma 2.3.10. *If $v \in V_i$, then $\Pr(\mathcal{E}(u)) \geq \frac{1}{4e^2}$.*

Proof. $\Pr(\mathcal{E}(u, v)) \geq p_i(1 - p_i)^{\deg_{G'}(u) + \deg_{G'}(v)} \geq p_i(1 - p_i)^{2d/2^{i-1}} \geq p_i e^{-2}$. Since $\deg_{G'}(u) \geq d/2^i$, $Pr(\mathcal{E}(u)) \geq \frac{d}{2^i} p_i e^{-2} \geq \frac{1}{4e^2}$ □

Therefore, if $v \in G_{\mathcal{F}} \setminus \Gamma^+(S)$ at the beginning of iteration l, the probability that $v \in \Gamma^+(S)$ at the end of iteration l is at least $1/(4e^2)$. We say a vertex in $G_{\mathcal{F}}$ *fails* if, after all $t = 4e^2 \ln(2e(d+1)^4)$ iterations, it is still not in $\Gamma^+(S)$.

Lemma 2.3.11. *Let S be an independent set selected by Weak-MIS. If $v \in \mathcal{F}$ then $\Pr(\Gamma^+(v) \cap S = \emptyset) \leq \frac{1}{2e(d+1)^4}$.*

Proof. By Lemma 2.3.10, the probability that v survives iteration ℓ conditioned on it surviving iterations 1 through $\ell - 1$ is at most $1 - 1/(4e^2)$. Over $t = 4e^2 \ln(2e(d+1)^4)$ iterations the probability of failure is at most $(1 - 1/(4e^2))^t \leq e^{-\ln(2e(d+1)^4)} = \frac{1}{2e(d+1)^4}$. □

The next step is to relate the number of rounds of Weak-MIS resampling with the size of witness trees.

Lemma 2.3.12. *Suppose a bad event is violated after k rounds of Weak-MIS resampling and the maximum depth of the witness trees is t, then there exists a sequence of not necessarily distinct vertices $v_1, \ldots, v_k$ such that the following hold:*

(1) $v_i \in G_i$, where G_i is the violated subgraph $G_{\mathcal{F}}$ the beginning of round i.

(2) $v_{i+1} \in \Gamma^+(v_i)$ for $1 \leq i \leq k - 1$.

(3) For at least $k - t$ indices $1 < l \leq k$, v_l failed in the call to Weak-MIS in round $l - 1$.

Proof. For $1 \leq i \leq k$, let S_i be the segment of the record C corresponding to events resampled at round i. Suppose that an event A is violated after k resampling rounds. Build a witness tree τ with root labeled A, adding nodes in the usual fashion, by scanning the record C in time-reversed order. For each j, in decreasing order, attach a node labelled $C(j)$ to the deepest node in τ whose label is in $\Gamma^+(C(j))$, if such a node in τ exists. Let $v_{k+1} = A$. We will build $v_k, v_{k-1}, \ldots, v_1$ in backward manner. For $k \geq i \geq 1$, we claim there is an event $v_i \in \Gamma^+(v_{i+1})$ such that either $v_i \in S_i$ or $v_i \in G_i$ and v_i failed at round i. If $v_{i+1} \notin G_i$ is not violated at the beginning of round i, then it must be the case that there exists an event $v_i \in \Gamma^+(v_{i+1})$ resampled at round i to cause $v_{i+1} \in G_{i+1}$. On the other hand, if $v_{i+1} \in G_i$ *is* violated at the beginning of round i, then either there exists $v_i \in \Gamma^+(v_{i+1})$ resampled at round i or v_{i+1} failed at round i. In the latter case, we let $v_i = v_{i+1}$. Notice that τ (excluding its artificial root labeled A) is a witness that occured and thus has depth at most t. Since in each of the k rounds, either the depth of our witness tree grows or a vertex fails, at least $k - t$ vertices must have failed in their respective rounds. □

Notice that the total possible number of sequences satisfying (2) in Lemma 2.3.12 is at most $n(d+1)^{k-1}$. Given a sequence of vertices $P = (v_1, \ldots, v_k)$ satisfying (2), define $X_P^{(i)}$ to be 1 if $v_i \in G_i$ and v_i failed, 0 otherwise. Let $X_P = \sum_{i=1}^{k} X_P^{(i)}$. If a sequence satisfying (1–3) occured, then there exists P such that $X_P \geq k - t$. Since $X_P^{(1)}, \ldots, X_P^{(i-1)}$ are determined by $S_1, \ldots, S_{i-1}$ and $G_1, \ldots, G_{i-1}$, $\mathrm{E}(X_P^{(i)} \mid X_P^{(1)}, \ldots, X_P^{(i-1)}) = \mathrm{E}(X_P^{(i)} \mid S_1, \ldots, S_{i-1}, G_1, \ldots, G_{i-1}) \leq q \stackrel{\text{def}}{=} \frac{1}{2e(d+1)^4}$ by Lemma 2.3.11. Fixing $t = k/2$, we have $k - t = k/2 = kq \cdot e(d+1)^4 \leq \mathrm{E}[X_P] \cdot e(d+1)^4$. By Lemma A.5 (Conditional Chernoff Bound):

$$\begin{aligned}\Pr(X_P \geq k/2) &\leq \left(\frac{e^{e(d+1)^4-1}}{(e(d+1)^4)^{e(d+1)^4}}\right)^{\frac{k}{2e(d+1)^4}} \\ &\leq \left(\frac{1}{(d+1)^2}\right)^k.\end{aligned}$$

By the union bound over all possible P satsfying (2), the probability that any such sequence

in Lemma 2.3.12 occurs is at most

$$n\,(d+1)^{k-1} \cdot \left(\frac{1}{(d+1)^2}\right)^k \leq n \cdot \left(\frac{1}{d+1}\right)^k.$$

Moser and Tardos showed that the probability that any witness tree of size at least t occurs is at most $(1-\epsilon)^t \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)}$. Thus, either a witness tree of depth at least $t = k/2$ occurs or there exists a sequence of vertices (as in Lemma 2.3.12) such that $t - k = k/2$ of them failed. The probability either of these occurs is at most $n \cdot \left(\frac{1}{d+1}\right)^k + (1-\epsilon)^{k/2} \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)}$ by the union bound.

2.3.3 A Sublogarithmic Algorithm

We have seen a faster algorithm for LLL when the general condition $ep(d+1) < 1$ is replaced by a stronger condition $p \cdot f(d) < 1$, where $f(d)$ is a faster growing function than $e(d+1)$. The question of how fast we can do for a stronger condition arises. Does there exist a sublogarithmic algorithm for faster growing $f(d)$, independent of n? We answer this affirmatively for an exponential function of d.

Inspired by [4], our approach is a two-stage approach. In the first stage, we run Algorithm 2 for $k(n)$ rounds. Then we identify the *dangerous* events, who are likely to become violated if some subset of its neighborhood is resampled. We will show there is a feasible solution by re-assigning the variables belonging to dangerous events. Moreover, we show the components induced by the dangerous events are likely to have *weak diameter* at most $k(n)$. The weak diameter of a component is the maximum distance w.r.t. the original graph of any pair in the component. In the second stage, each component of dangerous events computes the answer independent of others in time proportional to its weak diameter.

Consider an event A. Let $P_1(A), P_2(A)$ be probabilities such that $P_1(A)P_2(A) = 2^d \cdot \Pr(A)$. Given an assignment of the random variables, we say A is dangerous w.r.t. the current assignment if resampling of some subset of neighbors causes A to become violated with probability more than $P_2(A)$. We will show that the probability for A to become dangerous is at most $P_1(A)$.

Given that $P_2(A)$ is small enough for all $A \in \mathcal{A}$, we can find a feasible solution by re-assigning

the variables belonging to the dangerous vertices. Also, given that $P_1(A)$ is small enough, we will show that the weak diameter of each component after the first stage is at most k w.h.p. We explain the idea roughly. If we build a 2-witness tree rooted at a dangerous vertex after the first stage, the 2-witness tree has size at least k. If there exists a path consisting of dangerous vertices of length k after the first stage, we will show the union of the witness trees rooted at these vertices has size at least $\Omega(k \log k)$. Then, we will glue them together into a 3-witness tree. By choosing $k = \Theta(\log n / \log\log n)$, we would have a 3-witness tree with size $\Theta(\log n)$, which does not occur w.h.p.

Theorem 2.3.13 (Asymmetric LLL)**.** *Let* $\Pr(A) \le P_2(A) \le 1$ *and* $P_1(A) = 2^d \cdot \frac{\Pr(A)}{P_2(A)}$*, where* d *is the maximum degree of the dependency graph. If there exists an assignments of reals* $x_1, x_2 : \mathcal{A} \to (0, 0.99]$ *such that for all* $A \in \mathcal{A}$

1. $P_1(A) \le (1-\epsilon) x_1(A) \prod_{B \in \Gamma^3(A)} (1 - x_1(B))$
2. $P_2(A) \le x_2(A) \prod_{B \in \Gamma(A)} (1 - x_2(B))$

then the LLL problem can be solved in $O\left(\log_{1/(1-\epsilon)} n / \log\log_{1/(1-\epsilon)} n\right)$ *rounds.*

Corollary 2.3.14 (Symmetric LLL)**.** *Suppose that for all* $A \in \mathcal{A}$, $\Pr(A) \le p$ *and* A *shares variables with at most* d *other events in* $\mathcal{A}$*. Let* $z = 4ep2^d d^4$*. If* $z < 1$*, then a satisfying assignment can be found in* $O(\log_{1/z} n / \log\log_{1/z} n)$ *rounds.*

Proof of Collorary 2.3.14. For each $A \in \mathcal{A}$, let $P_2(A) = \frac{1}{4d} \ge p \ge \Pr(A)$ and so $P_1(A) = 2^d \cdot \frac{\Pr(A)}{P_2(A)} \le 4pd2^d$. Let $x_1(A) = 1/d^3$, $x_2(A) = 1/(2d)$ and $1 - \epsilon = 4ep2^d d^4$. First, we check that condition 1 in Theorem 2.3.13 holds

$$\begin{aligned}
(1-\epsilon) x_1(A) \prod_{B \in \Gamma^3(A)} (1 - x_1(A)) &= 4ep2^d d^4 \cdot \frac{1}{ed^3} \cdot \left(1 - \frac{1}{d^3}\right)^{|\Gamma^3(A)|} \\
&\ge 4ep2^d d \left(1 - \frac{1}{d^3}\right)^{d^3 - 1} \\
&\ge 4p2^d d = \Pr(A).
\end{aligned}$$

Condition 2 also holds similarly,

$$x_2(A) \prod_{B \in \Gamma(A)} (1 - x_2(A)) \geq \frac{1}{2d} \cdot \left(1 - \frac{1}{2d}\right)^d$$
$$= \frac{1}{2d} \cdot \frac{1}{2} = P_2(A).$$

□

Proof Sketch of Theorem 2.3.13. Given an assignment of each variables, we will classify the vertices into *safe* vertices and *dangerous* vertices. An event A is safe if the probability A becomes violated when any subset of its neighbors resample is at most $P_2(A)$. In contrast, the dangerous vertices are those where there exists a subset of neighbors whose resampling will cause it to be violated with probability greater than $P_2(A)$.

Using conditional probability, we can bound the probability that a vertex becomes dangerous after a random sampling of $\mathrm{vbl}(A)$ by $P_1(A) = 2^d \Pr(A)/P_2(A)$ (Lemma 2.3.15). Using Cond. 1 in Theorem 2.3.13, we show in Lemma 2.3.16 that after we resample dangerous vertices using the simple distributed algorithm for k rounds, if there exists a dangerous component whose weak diameter is at least k, then a 3-witness tree of size $\Omega(k \log k)$ would occur. When $k = \Theta(\log n / \log \log n)$, a 3-witness tree of size $O(\log n)$ would occur, which happens with probability at most $1/\operatorname{poly}(n)$. Therefore, with high probability, after $O(\log n / \log \log n)$ rounds of resampling, the weak diameters of the dangerous components are bounded by $O(\log n / \log \log n)$. Finally, a feasible assignment for a dangerous component can be found in $O(\log n / \log \log n)$ rounds locally, independent of other dangerous components, which can be argued using Cond. 2 in Theorem 2.3.13 and the definition of dangerous vertices.

Proof of Theorem 2.3.13. Fix $\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)$, let $T_{\mathcal{D}}$ denote the set of assignments b for $\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D})$ such that $b \in T_{\mathcal{D}}$ iff when the variables in $\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D})$ are fixed to be equal to b, the probability A becomes violated after sampling variables in $\mathrm{vbl}(\mathcal{D})$ exceeds $P_2(A)$, that is,

$$T_{\mathcal{D}} = \{b \mid \Pr(A \mid \mathrm{vbl}(A) \setminus \ \mathrm{vbl}(\mathcal{D}) = b) > P_2(A)\}$$

Given an assignment of the variables of A, we call A "dangerous" if there exists $\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)$

such that $\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D}) \in T_{\mathcal{D}}$. Otherwise, A is "safe". Notice that if A is violated then A is also dangerous, if we choose $\mathcal{D} = \emptyset$.

Lemma 2.3.15. $\Pr(A$ *becomes dangerous after (re)sampling* $\mathrm{vbl}(A)) \leq P_1(A)$.

Proof. By the union bound over each subset of neighbors, the probability that A becomes dangerous after sampling or resampling variables in $\mathrm{vbl}(A)$ is at most

$$\begin{aligned}
\sum_{\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)} \Pr(\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D}) \in T_{\mathcal{D}}) &= \sum_{\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)} \sum_{b \in T_{\mathcal{D}}} \Pr(\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D}) = b) \\
&= \sum_{\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)} \sum_{b \in T_{\mathcal{D}}} \frac{\Pr(A \cap (\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D}) = b))}{\Pr(A \mid \mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D}) = b)} \\
&\leq \sum_{\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)} \sum_{b \in T_{\mathcal{D}}} \frac{\Pr(A \cap (\mathrm{vbl}(A) \setminus \mathrm{vbl}(\mathcal{D}) = b))}{P_2(A)} \\
&\leq \sum_{\emptyset \subseteq \mathcal{D} \subseteq \Gamma(A)} \frac{\Pr(A)}{P_2(A)} \\
&\leq 2^d \cdot \frac{\Pr(A)}{P_2(A)} = P_1(A).
\end{aligned}$$

□

Notice that if A is safe, then if we resample all the variables of the dangerous events, the probability that A becomes violated is at most $P_2(A)$, by the definition of safe. By the second condition in Theorem 2.3.13 and the standard asymmetric LLL, there exists a feasible solution by reassigning only the variables of the dangerous events.

Let $E' \subseteq E$ be the edges having at least one endpoint that is dangerous. Let G' be the graph induced by E'. Each component of G' can compute the feasible solution independent of other components. (It is tempting to consider the components induced by only the dangerous vertices. However, when such components C_1 and C_2 are both adjacent to a safe vertex u, we have to consider C_1 and C_2 simultaneously to find an assignment that does not cause u to occur.)

Next we will show that the weak diameter of each component in G' is bounded. Note that if the weak diameter of each component in G' is at most D, then each component can find the feasible solution in $O(D)$ time. Each vertex will first learn the topology up to distance D,

which is possible in the LOCAL model. Then the leader in each component (say the vertex with the smallest ID) computes the feasible solution locally and then broadcasts the solution back to other vertices in the component.

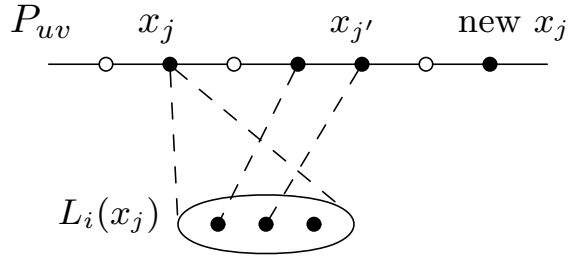
Lemma 2.3.16. *Suppose that the conditions in Theorem 2.3.13 hold, and there exists a component of weak diameter at least k. Then after running k rounds of the simple distributed algorithm, a 3-witness tree of size $\Omega(k \log k)$ occurs.*

Proof. Suppose that there exists u, v in the same component in G' and $\mathrm{dist}_G(u, v) = D \geq k$. Since u, v are connected in G', there exists a shortest u-v path P_{uv} of length at least D in G'. Notice that there are no consecutive safe vertices in P_{uv} by the definition of G'. Recall that S_i is the set of events resampled in round i. Let L_{k+1} be the set of dangerous vertices in P_{uv}. Ideally, one would build $|L_{k+1}|$ 2-witness trees of depth k, each rooted at each vertex in L_{k+1}, and then glue them together into a 3-witness tree of size $k \cdot |L_{k+1}|$. However, these 2-witness trees may overlap, so the final 3-witness tree may be much smaller. In the following, we will lower bound the size of the union of the 2-witness tree level by level and show that the size of the final 3-witness tree can be lower bounded.

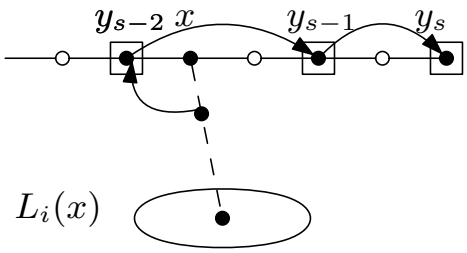
For each dangerous vertex x in P_{uv} (i.e. $x \in L_{k+1}$), define $L_{k+1}(x) = \{x\}$. For $1 \leq i \leq k$, define $L_i(x)$ inductively to be the set of events sampled during round i that are within distance 2 to any events in $L_{i+1}(x)$. Define $L_i = \bigcup_{x \in P_{uv}} L_i(x)$. For each $1 \leq i \leq k$, we will show the size of L_i is at least $\frac{D-2}{4(k-i+1)+2}$.

Notice that $L_i(x)$ must be non-empty, because by Lemma 2.3.2, for each $k+1 \geq j > i$ and each vertex w_j in L_j, there exists a vertex $w_{j-1} \in S_{j-1}$ such that $w_{j-1} \in \Gamma^{2+}(w_j)$. Also, for all $w \in L_i(x)$, $\mathrm{dist}_G(x, w) \leq 2(k - i + 1)$, since by definition of $L_i(x)$, there exists a sequence of vertices $(x = v_{k+1}, v_k, \ldots, v_i = w)$ such that $v'_i \in L_i(x)$ for $k+1 \geq i' \geq i$ and $\mathrm{dist}_G(v_{i'+1}, v_{i'}) \leq 2$ for $k+1 > i' \geq i$.

Let $P_{uv} = \{x_0, x_1, \ldots x_{|P_{uv}|}\}$. Let $j = 0$ if x_0 is dangerous; otherwise x_1 must be dangerous and we let $j = 1$. Repeat the following procedure (see Figure 1a): Select any $w \in L_i(x_j)$. Note that x_j must be dangerous and $L_i(x_j)$ is well-defined. Let $x_{j'}$ be the rightmost vertex in P_{uv} such that $w \in L_i(x'_j)$ (it can be the case that $j' = j$). If $x_{j'+1}$ is dangerous, set $j \leftarrow j' + 1$; otherwise $x_{j'+2}$ must be a dangerous vertex, then we set $j \leftarrow j' + 2$. Repeat until $j > |P_{uv}|$.



(a) An illustration of an iteration in the procedure for lower bounding L_i. The dashed lines are paths with length at most $2(k-i+1)$. In this iteration, the difference, Δ, between the new position and the old position of j is 5. Therefore, if $2 \cdot 2(k-i+1)+2 < 5$, then the detour from x_j to x'_j via $L_i(x_j)$ would be shorter the distance between x_j and x'_j on P_{uv}.



(b) An illustration showing that each resampled events in L_i is in the 3-witness tree rooted at y_s. The vertices inside the boxes are the independent set I. The dashed line is a sequence of vertices, where adjacent vertices have distance at most 2. The arrows links denote two vertices are within distance 3.

Figure 1

$|L_i|$ must be lower bounded by the total number of iterations l in the procedure above. We will show that we cannot move too far in each iteration, otherwise we would have a path shorter than $\text{dist}_G(u,v)$ connecting u and v. Let Δ_t be the difference of j at the beginning of iteration t and at the end of iteration t. The procedure terminates only if $\sum_{t=1}^{l} \Delta_t \geq |P_{uv}| - 2$ (The minus 2 came from the fact that the first and the last vertex in P_{uv} can be safe). Consider iteration t, if $\Delta_t > 4(k-i+1)+2$, it must reduce the distance between u and v by at least $\Delta_t - 4(k-i+1) - 2$. However, the total distance we can reduce is at most $|P_{uv}| - D$, for otherwise we would have a path connecting u and v with length less D, contradicting with $\text{dist}_G(u,v) = D$. Therefore,

$$\begin{aligned}
|P_{uv}| - D &\geq \sum_{t=1}^{l} (\Delta_t - 4(k-i+1) - 2) \\
&\geq (\sum_{t=1}^{l} \Delta_t) - (4(k-i+1)-2)\, l \\
&\geq |P_{uv}| - 2 - (4(k-i+1)-2)\, l
\end{aligned}$$

which implies

$$l \geq \frac{D-2}{4(k-i+1)-2} \geq \frac{k-2}{4(k-i+1)-2}.$$

Next, we will show that we can glue all the resampled events in $L_1, \ldots, L_k$ into a single 3-witness tree. We select an indepedent set $I = \{y_1, \ldots, y_s\} \subseteq L_{k+1}$ by starting from the leftmost vertex in L_{k+1} and repeatly selecting the first non-adjacent vertex in L_{k+1}. Therefore, y_{j+1} is in distance at most 3 from y_j for $1 \leq j < s$. Also, each $x_j \in L_{k+1}$ is adjacent to at least one vertex in I. Since I is an independent set, we can append $y_1, \ldots, y_s$ to our record artificially. We claim that each node in L_i for $1 \leq i \leq k$ corresponds to a node in the 3-witness tree rooted at y_s. For every node w in L_i, there must exist $x \in L_{k+1}$ such that $w \in L_i(x)$. Since x is adjacent to some $y_j \in I$, it implies w is in the 3-witness tree rooted at y_j. Finally, since y_j is a node in the 3-witness tree rooted at y_s, w must also be a node in the 3-witness tree rooted at y_s. The 3-witness tree rooted at y_s must have size at least $\sum_{i=1}^{k} \frac{k-2}{4(k-i+1)-2} = \Omega(k \log k)$. □

By chosing $k = \Omega\left(\frac{\log_{1/(1-\epsilon)} n}{\log\log_{1/(1-\epsilon)} n}\right)$, if there exists a component in G' with diameter at least k, then there exists a 3-witness of size at least $\Omega(\log_{1/(1-\epsilon)} n)$ w.h.p. However, by Condition 1 in Theorem 2.3.13 and by Lemma 2.3.6, the probability that such a 3-witness tree occurs is at most $1/\operatorname{poly}(n)$. Therefore, we can conclude that after $O\left(\frac{\log_{1/(1-\epsilon)} n}{\log\log_{1/(1-\epsilon)} n}\right)$ rounds, the weak diameter of each component in G' is at most $O\left(\frac{\log_{1/(1-\epsilon)} n}{\log\log_{1/(1-\epsilon)} n}\right)$ w.h.p. and the solution can be found in time proportional to the weak diameter. □

2.3.4 Lower Bound

Linial [118] proved that in an n-vertex ring, any distributed $(\log^{(k)} n)$-coloring algorithm requires $\Omega(k)$ rounds of communication, even if randomization is used. In particular, $O(1)$-coloring a ring requires $\Omega(\log^* n)$ time. We prove that Linial's lower bound implies that even weak versions of the Lovász Local Lemma cannot be computed in constant time.

Theorem 2.3.17. *Let $\mathcal{P}$, $\mathcal{A}$, and $G_{\mathcal{A}}$ be defined as usual. Let d be the maximum degree of any vertex in $G_{\mathcal{A}}$, $p = \max_{A \in \mathcal{A}} \Pr(A)$ be the maximum probability of any bad event, and*

$f : \mathbb{N} \to \mathbb{N}$ be an arbitrarily quickly growing function, where $f(d) \geq e(d+1)$. If $p \cdot f(d) < 1$ then $\Pr(\bigcap_{A \in \mathcal{A}} \overline{A}) > 0$. However, $\Omega(\log^* |\mathcal{A}|)$ rounds of communication are required for the vertices of $G_{\mathcal{A}}$ to agree on a point in $\bigcap_{A \in \mathcal{A}} \overline{A}$.

The purpose of the function f is to show that our lower bound is insensitive to significant weakening of the standard criterion "$ep(d+1) < 1$." We could just as easily substitute $e^{e^d} p < 1$ or any similar criterion, for example.

Proof. Consider the following coloring procedure. Each vertex in an n-vertex ring selects a color from $\{1, \ldots, c\}$ uniformly at random. An edge is *bad* if it is monochromatic, an event that holds with probability $p = 1/c$. Let $\mathcal{A}$ be the dependency graph for these events having maximum degree $d = 2$ and choose c to be (the constant) $f(2) + 1$, for any quickly growing function f. It follows from the LLL that a good c-coloring exists since $p \cdot f(2) < 1$. However, by [118], the vertices of $G_{\mathcal{A}}$ require $\Omega(\log^* n - \log^* c) = \Omega(\log^* n)$ time to find a good c-coloring. □

It is also possible to obtain *conditional* lower bounds on distributed versions of the LLL. For example, the best known randomized $O(\Delta)$-coloring algorithm takes $\exp(O(\sqrt{\log \log n}))$ time [11], though better bounds are possible if $\Delta \gg \log n$ [157]. If LLL could be solved in less than $\exp(O(\sqrt{\log \log n}))$ time then we could improve on [11], as follows. Each vertex in G selects a color from a palette of size $c \geq 2e\Delta$ uniformly at random. As usual, an edge is bad if it is monochromatic. The dependency graph of these bad events corresponds to the line graph of G, which has maximum degree $d = 2\Delta - 2$. Since $e(1/c)(d+1) < 1$, a valid coloring can be found with one invocation of an LLL algorithm.

2.4 Applications

The Lovász Local Lemma has applications in many coloring problems, such as list coloring, frugal coloring, total coloring, and coloring triangle-free graphs [125]. We give a few examples of constructing these colorings distributively. In these applications, the existential bounds are usually achieved by the so called "Rödl Nibble" method or the semi-random method. The method consists of one or more iterations. Each iteration is a random process and some local

properties are maintained in the graph. The properties depend on the randomness within a constant radius. Each property is associated with a bad event, which is the event that the property fails to hold. The Lovász Local Lemma can then be used to show the probability none of the bad events hold is positive, though it may be exponentially small in the size of the graph. This probability can then be amplified in a distributed fashion using a Moser-Tardos-type resampling algorithm. Notice that we will need to find an independent set (e.g., an MIS or Weak-MIS or set of events with locally minimal IDs) *in the dependency graph* induced by the violated local properties. Since we assumed the LOCAL model, the violated local properties can be identified in constant time and the algorithms for MIS/Weak-MIS can be simulated with a constant factor overhead, where each property is taken care by one of the processors nearby (within constant distance). The important point here is that the dependency graph and the underlying distributed network are sufficiently similar so that distributed algorithms on one topology can be simulated on the other with $O(1)$ slowdown.

Most applications of the LLL demand $epd^2 < 1$ or even weaker bounds. In this case, the efficient simple distributed algorithm can be applied. (The local properties are often that some quantities do not deviate too much from their expectations. Thus, the the failure probability of each local property is often bounded via standard Chernoff-type concentration inequalities.)

2.4.1 Distributed Defective Coloring

We begin with a simple single-iteration application that uses the local lemma. Let $\phi : V \to \{1, 2, \ldots, k\}$ be a k-coloring. Define $\mathrm{def}_\phi(v)$ to be the number of neighbors $w \in N(v)$ such that $\phi(v) = \phi(w)$. The coloring ϕ is said to be f-defective if $\max_v \mathrm{def}_\phi(v) \leq f$. Barenboim and Elkin ([9], Open Problem 10.7) raised the problem of devising an efficient distributed algorithm for computing an f-defective $O(\Delta/f)$-coloring.

To warm up, we give a simple procedure for obtaining an f-defective $O(\Delta/f)$-coloring in $O(\log n/f)$ time w.h.p., for $f \geq 60 \ln \Delta$. Suppose each vertex colors itself with a color selected from $\{1, 2, \ldots, \lceil 2\Delta/f \rceil\}$ uniformly at random. For every $v \in N(u)$, let X_v be 1 if v is colored the same as u, 0 otherwise. Let $X = \sum_{v \in N(u)} X_v$ denote the number of neighbors colored the same as v. Let A_u denote the bad event that $X > f$ at u. Clearly, whether A_u

occurs is locally checkable by u in a constant number of rounds. Moreover, the event A_u only depends on the the random choices of u's neighbors. If A_u occured and is selected for resampling, the colors chosen by u and its neighbors will be resampled. The dependency graph $G_{\mathcal{A}}$ has maximum degree $d = \Delta^2$, because two events share variables only if they are within distance two. Now we will calculate the probability that A_u occurs. If we expose the choice of u first, then $\Pr(X_v = 1) \leq f/(2\Delta)$ and it is independent among other $v \in N(u)$. Letting $M = f/2$, we have $E[X] \leq f/2 = M$. By Lemma A.4, $\Pr(X > f) \leq e^{-f/6}$. Let A_u denote the bad event that $X > f$ at u. Therefore, $epd^2 \leq e^{-(f/6-1-4\ln\Delta)} \leq e^{-(f/12)}$, since $f \geq 60\ln\Delta$. By using the simple distributed algorithm, it takes $O(\log_{1/epd^2} n) = O(\log n/f)$ rounds to avoid the bad events w.h.p.

Next, we show that there is a constant $C > 0$ such that for any $f \geq C$, an f-defective $O(\Delta/f)$-coloring can be obtained in $O(\log n/f)$ rounds. For $f < C$, we can use the $(\Delta+1)$-coloring algorithms to obtain 0-defective (proper) $(\Delta+1)$-colorings that runs in $O(\log n)$ rounds. Let $\Delta_0 = \Delta$ and $\Delta_i = \log^3 \Delta_{i-1}$.

if $f < 60\ln\Delta_{i-1}$ **then**
  Each node in G' chooses a color from $\lceil(1+6\Delta_i^{-1/3})\cdot\frac{\Delta_{i-1}}{\Delta_i}\rceil$ colors uniformly at random.
  Let A_u denote the event that more than Δ_i neighbors of u are colored the same with u.
  Run Algorithm 2 until no bad events A_u occurs.
  Let G_j denote the graph induced by vertices with color j.
  For $j = 1\ldots, \lceil(1+6\Delta_i^{-1/3})\cdot\frac{\Delta_{i-1}}{\Delta_i}\rceil$, call defective-coloring(G_j, $i+1$) in parellel.
else
  Obtain an f-defective, $(2\Delta_{i-1}/f)$-coloring for G'.
end if

Algorithm 4: defective-coloring(G', i)

An f-defective $O(\Delta/f)$-coloring in G can be obtained by calling defective-coloring(G, 1), which is described in Algorithm 4. The procedure defective-coloring(G', i) is a recurisve procedure whose halting condition is when $f \geq 60\log\Delta_{i-1}$. When the condition occurs, we will use the procedure described above to obtain an f-defective $(2\Delta_{i-1}/f)$-coloring in G'. Let l denote the total number of levels of the recursion. The final color of node v is a vector $(c_1, c_2, \ldots, c_l)$, where c_i denotes the color received by v at level i. Clearly, such a coloring obtained by the procedure is f-defective. The total number of colors used is:

$$\left(\prod_{1\le i<l}(1+6\Delta_i^{-1/3})\cdot\frac{\Delta_{i-1}}{\Delta_i}\right)\cdot\frac{2\Delta_{l-1}}{f}=2(\Delta/f)\cdot\prod_{1\le i<l}\left(1+\frac{6}{\log\underbrace{\log^3\ldots\log^3\Delta}_{i-1}}\right)=O(\Delta/f).$$

Now we will analyze the number of rounds needed in each level i. Suppose that each vertex colors itself with a color selected from $\{1,2,\ldots,\lceil(1+6\Delta_i^{-1/3})\cdot\frac{\Delta_{i-1}}{\Delta_i}\rceil\}$ uniformly at random. For every $v\in N(u)$, let X_v be 1 if v is colored the same as u, 0 otherwise. Let $X=\sum_{v\in N_{G'}(u)}X_v$ denote the number of neighbors colored the same as v. Let A_u denote the bad event that $X>\Delta_i$ at u. The dependency graph $G_{\mathcal{A}}$ has maximum degree $d=\Delta_{i-1}^2$, because two events share variables only if they are within distance two. If we expose the choice of u first, then $\Pr(X_v=1)\le\frac{\Delta_i}{\Delta_{i-1}}\cdot\frac{1}{1+6\Delta_i^{-1/3}}$ and it is independent among other $v\in N_{G'}(u)$. Since the maximum degree of G' is Δ_{i-1}, $\mathrm{E}[X]\le\Delta_i\cdot\frac{1}{1+6\Delta_i^{-1/3}}$. By Chernoff Bound (Lemma A.4),

$$\Pr(A_u)=\Pr(X>\Delta_i)\le\Pr(X>(1+6\Delta_i^{-1/3})\cdot\mathrm{E}[X])\le e^{-6^2\Delta_i^{-2/3}\cdot\mathrm{E}[X]/3}\le e^{-6\Delta_i^{1/3}}=e^{-6\ln\Delta_{i-1}}.$$

Therefore, $epd^2\le e^{-\ln\Delta_{i-1}}$ and so Algorithm 2 runs in $O(\log n/\log\Delta_{i-1})$ rounds. The total number of rounds over all levels is therefore

$$O\left(\log n\cdot\left(\frac{1}{\log\Delta}+\frac{1}{\log\log^3\Delta}+\ldots+\frac{1}{\log\Delta_{l-1}}+\frac{1}{f}\right)\right)=O\left(\frac{\log n}{f}\right).$$

2.4.2 Distributed Frugal Coloring

A *β-frugal coloring* of a graph G is a proper vertex-coloring of G such that no color appears more than β times in any neighborhood. Molloy and Reed [125] showed the following by using an asymmetric version of the local lemma:

Theorem 2.4.1. *For any constant integer $\beta\ge1$, if G has maximum degree $\Delta\ge\beta^\beta$ then G has a β-frugal proper vertex coloring using at most $16\Delta^{1+\frac{1}{\beta}}$ colors.*

Here we outline their proof and show how to turn it into a distributed algorithm that finds such a coloring in $O(\log n\cdot\log^2\Delta)$ rounds. If $\beta=1$, then simply consider the square

graph of G, which is obtained by adding the edges between vertices whose distance is 2. A proper coloring in the square graph is a 1-frugal coloring in G. Since the square graph has maximum degree Δ^2, it can be (Δ^2+1)-colored by simulating distributed algorithms for $(\Delta+1)$-coloring.

For $\beta \geq 2$, let $k = 16\Delta^{1+\frac{1}{\beta}}$. Suppose that each vertex colors itself with one of the k colors uniformly at random. Consider two types of bad events. For each edge uv, the Type I event $A_{u,v}$ denotes that u and v are colored the same. For each subset $\{u_1, \ldots, u_{\beta+1}\}$ of the neighborhood of a vertex, Type II event $A_{u_1,\ldots,u_{\beta+1}}$ denotes that $u_1, \ldots, u_{\beta+1}$ are colored the same. If none of the events occur, then the random coloring is a β-frugal coloring. For each Type I event $A_{u,v}$, $\Pr(A_{u,v})$ is at most $1/k$. For each Type II event $A_{u_1,\ldots,u_{\beta+1}}$, $\Pr(A_{u_1,\ldots,u_{\beta+1}}) \leq 1/k^\beta$. For each bad event A, let $x(A) = 2\Pr(A)$. Notice that $x(A) \leq 1/2$, we have:

$$\begin{aligned} x(A) \prod_{B\in\Gamma(A)} (1-x(B)) &\geq x(A) \prod_{B\in\Gamma(A)} \exp\left(-x(B)\cdot 2\ln 2\right) \quad \{(1-x) \geq e^{-x\cdot 2\ln 2} \text{ for } x \leq 1/2\} \\ &= x(A)\cdot \exp\left(-2\ln 2 \cdot \sum_{B\in\Gamma(A)} 2\Pr(B)\right) \end{aligned}$$

Since A shares variables with at most $(\beta+1)\Delta$ Type I events and $(\beta+1)\Delta\binom{\Delta}{\beta}$ Type II events,

$$\begin{aligned} \sum_{B\in\Gamma(A)} \Pr(B) &\leq (\beta+1)\Delta\cdot\frac{1}{k} + (\beta+1)\Delta\binom{\Delta}{\beta}\cdot\frac{1}{k^\beta} \\ &< \frac{(\beta+1)\Delta}{k} + \frac{(\beta+1)\Delta^{\beta+1}}{\beta! k^\beta} \\ &= \frac{\beta+1}{16\Delta^{\frac{1}{\beta}}} + \frac{\beta+1}{\beta!(16)^\beta} \\ &< 1/8 \qquad \text{(for } \Delta \geq \beta^\beta \text{ and } \beta \geq 2\text{)} \end{aligned}$$

Therefore,

$$x(A)\prod_{B\in\Gamma(A)}(1-x(B)) \geq x(A)\exp\left(-\frac{\ln 2}{2}\right)$$
$$= \sqrt{2}\cdot \Pr(A).$$

By letting $1-\epsilon = 1/\sqrt{2}$ in Theorem 2.3.8, we need at most $O(\log_{\sqrt{2}} n)$ rounds of weak MIS resampling. In each resampling round, we have to identify the bad events first. Type I events $A_{u,v}$ can be identified by either u or v in constant rounds, where ties can be broken by letting the node with smaller ID check it. If $\{u_1,\ldots,u_{\beta+1}\}$ is in the neighborhood of u, then the Type II event $A_{u_1,\ldots,u_{\beta+1}}$ will be checked by u. If $\{u_1,\ldots,u_{\beta+1}\}$ is in the neighborhood of multiple nodes, we can break ties by letting the one having the smallest ID to check it. All Type II events in the neighborhood of u can be identified from the colors selected by the neighbors of u. Next we will find a weak MIS induced by the bad events in the dependency graph. Each node will simulate the weak MIS algorithm on the events it is responsible to check. Each round of the weak MIS algorithm in the dependency graph can be simulated with constant rounds. The maximum degree d of the dependency graph is $O((\beta+1)\Delta\binom{\Delta}{\beta})$. Therefore, we need at most $O(\log n\cdot\log^2 d) = O(\log n\cdot\log^2\Delta)$ rounds, since β is a constant and $(\beta+1)\Delta\binom{\Delta}{\beta} \leq (\beta+1)\Delta^{\beta+1} = \text{poly}(\Delta)$.

β-frugal, $(\Delta+1)$-coloring

The frugal $(\Delta+1)$-coloring problem for general graphs is studied by Hind, Molloy, and Reed [81], Pemmaraju and Srinivasan [145], and Molloy and Reed [126]. In particular, the last one gave an upper bound of $O(\log\Delta/\log\log\Delta)$ on the frugality of $(\Delta+1)$-coloring. This is optimal up to a constant factor, because it matches the lower bound of $\Omega(\log\Delta/\log\log\Delta)$ given by Hind et al. [126]. However, it is not obvious whether it can be implemented efficiently in a distributed fashion, because they used a structural decomposition computed by a sequential algorithm. Pemmaraju and Srinivasan [145] showed an existential upper bound of $O(\log^2\Delta/\log\log\Delta)$. Furthermore, they gave a distributed algorithm that computes an $O(\log\Delta\cdot\frac{\log n}{\log\log n})$-frugal, $(\Delta+1)$-coloring in $O(\log n)$ rounds. We show how to improve it to find a $O(\log^2\Delta/\log\log\Delta)$-frugal, $(\Delta+1)$-coloring also in $O(\log n)$ rounds.

They proved the following theorem:

Theorem 2.4.2. *Let G be a graph with maximum vertex degree Δ. Suppose that associated with each vertex $v \in V$, there is a palette $P(v)$ of colors, where $|P(v)| \geq \deg(v) + 1$. Furthermore, suppose $|P(v)| \geq \Delta/4$ for all vertices v in G. Then, for some subset $C \subseteq V$, there is a list coloring of the vertices in C such that:*

(a) $G[C]$ is properly colored.

(b) For every vertex $v \in V$ and for every color x, there are at most $9 \cdot \frac{\ln \Delta}{\ln \ln \Delta}$ neighbors of v colored x.

(c) For every vertex $v \in V$, the number of neighbors of v not in C is at most $\Delta(1 - \frac{1}{e^5}) + 27\sqrt{\Delta \ln \Delta}$.

(d) For every vertex $v \in V$, the number of neighbors of v in C is at most $\frac{\Delta}{e^5} + 27\sqrt{\Delta \ln \Delta}$.

The theorem was obtained by applying the LLL to the following random process: Suppose that each vertex v has an unique ID. Every vertex picks a color uniformly at random from its palette. If v has picked a color that is not picked by any of its neighbor whose ID is smaller than v, then v will be colored with that color. Let q_v denote the probability that v becomes colored. Then, if v is colored, with probability $1 - 1/(e^5 q_v)$, v uncolors itself. This ensures that the probability that v becomes colored in the process is exactly $1/e^5$, provided that $q_v \geq 1/e^5$, which they have shown to be true.

They showed by iteratively applying the theorem for $O(\log \Delta)$ iterations, an $O(\log^2 \Delta / \log \log \Delta)$-frugal, $(\Delta + 1)$-coloring can be obtained. Let G_i be the graph after round i obtained by deleting already colored vertices and Δ_i be the maximum degree of G_i. The palette $P(u)$ for each vertex u contains colors that have not been used by its neighbors. It is always true that $|P(v)| \geq \deg(v) + 1$. Notice that to apply Theorem 2.4.2, we also need the condition $|P(v)| \geq \Delta/4$. The worst case behavior of Δ_i and p_i is captured by the recurrences:

$$\begin{aligned} \Delta_{i+1} &= \Delta_i \left(1 - \frac{1}{e^5}\right) + 27\sqrt{\Delta_i \ln \Delta_i} \\ p_{i+1} &= p_i - \frac{\Delta_i}{e^5} - 27\sqrt{\Delta_i \ln \Delta_i}. \end{aligned} \tag{2.1}$$

They showed the above recurrence can be solved to obtain the following bounds on Δ_i and p_i:

Lemma 2.4.3. *Let $\alpha = (1 - 1/e^5)$. There is a constant C such that for all i for which $\Delta_i \geq C$, $\Delta_i \leq 2\Delta_0\alpha^i$ and $p_i \geq \frac{\Delta_0}{2}\alpha^i$.*

Therefore, $|P(v)| \geq \Delta/4$ always holds. The two assumptions of Theorem 2.4.2 are always satisfied and so it can be applied iteratively until $\Delta_i < C$, which takes at most $\log_{1/\alpha}\left(\frac{2\Delta_0}{C}\right) = O(\log \Delta)$ iterations. Since each iteration introduces at most $O(\log \Delta / \log\log \Delta)$ neighbors of the same color to each vertex, the frugality will be at most $O(\log^2 \Delta / \log\log \Delta)$. In the end, when $\Delta_i < C$, one can color the remaining graph in $O(\Delta_i + \log^* n)$ time using existing $(\Delta_i + 1)$-coloring algorithms [10]. This will only add $O(1)$ copies of each color to the neighborhood, yielding a $O(\log^2 \Delta / \log\log \Delta)$-frugal, $(\Delta + 1)$-coloring. In order to make it suitable for our simple distributed algorithm and achieve the running time of $O(\log n)$, we will relax the criteria of (b),(c),(d) in Theorem 2.4.2:

(b') For every vertex $v \in V$ and for every color x, there are at most $18 \cdot \frac{\ln \Delta_0}{\ln\ln \Delta_0}$ neighbors of v colored x.

(c') For every vertex $v \in V$, the number of neighbors of v not in C is at most $\Delta(1 - \frac{1}{e^5}) + 40\sqrt{\Delta}\ln \Delta$.

(d') For every vertex $v \in V$, the number of neighbors of v in C is at most $\frac{\Delta}{e^5} + 40\sqrt{\Delta}\ln \Delta$.

In (b'), Δ is replaced by Δ_0, which is the maximum degree of the initial graph. Also, the constant 9 is replaced by 18. In (c') and (d'), the constant 27 is replaced by 40 and $\sqrt{\ln \Delta}$ is replaced by $\ln \Delta$. It is not hard to see that Lemma 2.4.3 still holds and an $O(\log^2 \Delta / \log\log \Delta)$-frugal coloring is still obtainable. Originally, by Chernoff Bound and Azuma's Inequality, they showed

$$\Pr\left(\#\text{ neighbors of } v \text{ colored } x \text{ exceeds } 9 \cdot \tfrac{\ln \Delta}{\ln\ln \Delta}\ \right) < \frac{1}{\Delta^6} \tag{2.2}$$

and

$$\Pr\left(\left|P_v - \frac{\deg(v)}{e^5}\right| > 27\sqrt{\Delta \ln \Delta}\right) < \frac{2}{\Delta^{4.5}} \tag{2.3}$$

where P_v is the number of colored neighbors of v. Theorem 2.4.2 can be derived from (2.2) and (2.3). The relaxed version (b'), (c'), and (d') can be shown to fail with a lower probability.

$$\Pr\left(\#\text{ neighbors of } v \text{ colored } x \text{ exceeds } 18 \cdot \tfrac{\ln \Delta_0}{\ln \ln \Delta_0}\ \right) < \frac{1}{\Delta_0^{12}} \tag{2.4}$$

and

$$\Pr\left(\left|P_v - \frac{\deg(v)}{e^5}\right| > 40\sqrt{\Delta}\ln\Delta\right) < \frac{2}{\Delta^{9\ln\Delta}} \tag{2.5}$$

The bad event A_v is when the neighbors of v colored x exceeds $18 \cdot \frac{\ln \Delta_0}{\ln \ln \Delta_0}$ for some color x or $|P_v - \frac{\deg(v)}{e^5}| > 40\sqrt{\Delta}\ln\Delta$ happens. By (2.4), (2.5), and the union bound, $\Pr(A_v) \leq (\Delta+1)/\Delta_0^{12} + 2/\Delta^{9\ln\Delta}$. In their random process, they showed A_v depends on variables up to distance two. Thus, the dependency graph $G_{\mathcal{A}}$ has maximum degree d less than Δ^4. Note that

$$\begin{aligned} epd^2 &= e\Delta^8((\Delta+1)/(2\Delta_0^{12}) + 2/\Delta^{9\ln\Delta}) \\ &\leq 1/(2\Delta_0) + 1/(2\Delta^{\ln\Delta}) \\ &< 2 \cdot \max(1/(2\Delta_0), 1/(2\Delta^{\ln\Delta})) \\ &= \max(1/\Delta_0, 1/\Delta^{\ln\Delta}). \end{aligned}$$

The number of resampling rounds needed is at most $O(\log_{\frac{1}{epd^2}} n)$, which is at most $\frac{\ln n}{\min(\ln\Delta_0, \ln^2\Delta)} \leq \frac{\ln n}{\ln\Delta_0} + \frac{\ln n}{\ln^2\Delta}$. Therefore, the total number of rounds needed is at most:

$$\begin{aligned} &\sum_{i=1}^{c\ln\Delta_0}\left(\frac{\ln n}{\ln\Delta_0} + \frac{\ln n}{\ln^2\Delta_i}\right) \\ &\leq \sum_{i=1}^{c\ln\Delta_0}\left(\frac{\ln n}{\ln\Delta_0} + \frac{\ln n}{\ln^2(2\Delta_0\alpha^i)}\right) \\ &= c\ln\Delta_0 \cdot \frac{\ln n}{\ln\Delta_0} + \ln n \sum_{i=1}^{c\ln\Delta_0} \frac{1}{(\ln\Delta_0 - i\ln\frac{1}{\alpha} + \ln 2)^2} \\ &\leq c\ln n + \ln n \cdot O\left(\sum_{i=1}^{\infty}\frac{1}{i^2}\right) = O(\log n) \end{aligned}$$

where $c > 0$ is some constant, and $\alpha = (1 - 1/e^5)$.

2.4.3 Distributed List Coloring

Given a graph G, each vertex v is associated with a list (or a *palette*) of available colors $P(v)$. Let $\deg_c(v)$ denote the number of neigbors $w \in N(v)$ such that c is $P(w)$. Suppose that $\deg_c(v)$ is upper bounded by D. The list coloring constant is the minimum K such that for any graph G and any palettes $P(u)$ for $u \in G$, if $|P(u)| \geq K \cdot D$ and $\deg_c(u) \leq D$ for every $u \in G$ and every $c \in P(u)$, then a proper coloring can be obtained by assigning each vertex a color from its list. Reed [152] first showed the list coloring constant is at most $2e$ by a single application of LLL. Haxell [80] showed 2 is sufficient. Later, Reed and Sudakov [153] used a multiple iterations Rödl Nibble method to show the list coloring constant is at most $1 + o(1)$, where $o(1)$ is a function of D. Reed's upper bound of $2e$ can be made distributed and constructive with a slighty larger factor, say $2e + \epsilon$ for any constant $\epsilon > 0$. The LLL condition they need is close to tight and so we will need to use the weak MIS algorithm. The additional slack needed is due to the ϵ-slack needed in distributed LLL ($ep(d+1) \leq 1 - \epsilon$). The constructive algorithm can be easily transformed from their proof. Here we outline their proof: Suppose $|P(v)| \geq (2e + \epsilon)D$ for all v. Each vertex is assigned a color from its palette uniformly at random. They showed that with positive probability, a proper coloring is obtained. Let $e = uv \in E$, and $c \in P(u) \cap P(v)$. Define $A_{e,c}$ to be the bad event that both u and v are assigned c. Clearly, $p = \Pr(A_{e,c}) = 1/((2e+\epsilon)D)^2$. Also, there are at most $(2e+\epsilon)D^2$ events that depend on the color u picks and at most $(2e+\epsilon)D^2$ events that depend on the color v picks. The dependency graph has maximum degree $d = 2(2e+\epsilon)D^2 - 2$. Since $ep(d+1) \leq 2e/(2e+\epsilon)$ is upper bounded by a constant less than 1, we can construct the coloring in $O(\log n \cdot \log^2 D)$ rounds by using the weak MIS algorithm.

In the following, we shall show that for any constants $\epsilon, \gamma > 0$, there exists $D_{\epsilon,\gamma} > 0$ such that for any $D \geq D_{\epsilon,\gamma}$, any $(1+\epsilon)D$-list coloring instance can be colored in $O(\log^* D \cdot \max(1, \log n/D^{1-\gamma}))$ rounds. The algorithm consists of multiple iterations. Let $P_i(u)$ and $\deg_{i,c}(u)$ be the palette and the c-degree of u at end of iteration i. Also, at the end of iteration i, denote the neighbor of u by $N_i(u)$ and the c-neighbor by $N_{i,c}(u)$, which are the neighbors of u having c in their palette. Suppose that each vertex u has an unique ID, $\mathrm{ID}(u)$. Let $N^*_{i,c}(u)$ denote the set of c-neighbors at the end of iteration i having smaller ID than u. Let $\deg^*_{i,c}(u) = |N^*_{i,c}(u)|$.

In each iteration i, each vertex will select a set of colors $S_i(u) \subseteq P_{i-1}(u)$ and $K_i(u) \subseteq P_{i-1}(u)$,

List-Coloring $(G, \{\pi_i\}, \{\beta_i\})$

$G_0 \leftarrow G$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  for each $u \in G_{i-1}$ **do**
    $(S_i(u), K_i(u)) \leftarrow \text{Select}(u, \pi_i, \beta_i)$
    Set $P_i(u) \leftarrow K_i(u) \setminus S_i(N^*_{i-1}(u))$
    if $S_i(u) \cap P_i(u) \neq \emptyset$ **then** color u with any color in $S_i(u) \cap P_i(u)$ **end if**
  end for
  $G_i \leftarrow G_{i-1} \setminus \{\text{colored vertices}\}$
until

Algorithm 5

Select(u, π_i, β_i)

Include each $c \in P_{i-1}(u)$ in $S_i(u)$ independently with probability π_i.
For each c, calculate $r_c = \beta_i/(1 - \pi_i)^{\deg^*_{i-1,c}(u)}$.
Include $c \in P_{i-1}(u)$ in $K_i(u)$ independently with probability r_c.
return $(S_i(u), K_i(u))$.

Algorithm 6

which are obtained from Algorithm 6. If a color is in $K_i(u)$ and it is not in $S_i(v)$ for any $v \in N^*_{i-1}(u)$, then it remains in its new palette $P_i(u)$. Furthermore, if $S_i(u)$ contains a color that is in $P_i(u)$, then u colors itself with the color (in case there are multiple such colors, break ties arbitrarily).

Given π_i, the selecting probability for each vertex u to include a color in $S_i(u)$, the probability that $u \notin S_i(N^*_{i-1}(u))$ is $(1-\pi_i)^{\deg^*_{i-1,c}(u)}$. Define $\beta_i = (1-\pi_i)^{t'_{i-1}}$, where t'_{i-1} is an upper bound on $\deg_{i-1,c}(u)$ for each vertex u and each color c. Then $r_c = \beta_i/(1-\pi_i)^{\deg^*_{i-1,c}(u)}$ is always at most 1 and thus it is a valid probability. Therefore, the probability that a color $c \in P_{i-1}(u)$ remains in $P_i(u)$ is $(1-\pi_i)^{\deg^*_{i-1,c}(u)} \cdot r_c = \beta_i$. As a result, the palette size shrinks by at most a β_i factor in expectation.

Suppose that p'_i is the lower bound on the palette size at the end of iteration i. Then the probability that u remains uncolored is upper bounded by the probability that any of the colors in $P_i(u)$ was not seleted to be in $S_i(u)$. The probability is *roughly* $(1-\pi_i)^{p'_i}$, which we will define it to be α_i. The slight inaccuracy comes from the fact that we are conditioning on the new palette size $|P_i(u)|$ is lower bounded by p'_i. However, we will show the effect of this conditioning only affects the probability by a small amount.

Let $p_0 = (1+\epsilon) \cdot D$ and $t_0 = D$ be the initial palette size and upper bound on c-degree. In the following, p_i and t_i are the ideal lower bound of the palette size and the ideal upper bound of the c-degree at the end of each iteration i. p'_i and t'_i are the approximation of p_i and t_i, incoporating the errors from concentration bounds. K is a constant in the selecting probability that depends on ϵ. T is the threshold on the c-degree before we switch to a different analysis, since the usual concentration bound does not apply when the quantity is small. $\delta = 1/\log D$ is the error control parameter which is set to be small enough such that $(1 \pm \delta)^i$ is $1 \pm o(1)$ for every iteration i.

$$
\begin{aligned}
\pi_i &= 1/(Kt'_{i-1}+1) & \delta &= 1/\log D \\
\alpha_i &= (1-\pi_i)^{p'_i} & \beta_i &= (1-\pi_i)^{t'_{i-1}} \\
p_i &= \beta_i p_{i-1} & t_i &= \max(\alpha_i t_{i-1}, T) \\
p'_i &= (1-\delta)^i p_i & t'_i &= (1+\delta)^i t_i \\
K &= 2 + 2/\epsilon & T &= D^{1-0.9\gamma}/2
\end{aligned}
$$

Intuitvely, we would like to have t_i shrink faster than p_i. To ensure this happens, we must have $\alpha_1 \leq \beta_1$, which holds under our setting of π_i. As we will show, α_i shrinks much faster than β_i as i becomes larger. Note that β_i is at least a constant, as

$$\begin{aligned}
\beta_i &= (1 - 1/(Kt'_{i-1} + 1))^{t'_{i-1}} \\
&= (1 - 1/(Kt'_{i-1} + 1))^{(Kt'_{i-1}) \cdot (1/K)} \\
&\geq (e^{-1})^{1/K} = e^{-1/K} && \text{since } (1 - 1/(x+1))^x \geq e^{-1}.
\end{aligned}$$

Lemma 2.4.4. *$t_r = T$ after at most $r = O(\log^* D)$ iterations.*

Proof. We divide the iterations into two stages, where the first stage consists of iterations i for which $t_{i-1}/p_{i-1} \geq 1/(1.1e^{2/K}K)$. During the first stage, we show that the ratio t_i/p_i decreases by a factor of $\exp\left(-(1-o(1))\frac{\epsilon^2}{4(1+\epsilon)}\right)$ in every round.

$$\begin{aligned}
\frac{t_i}{p_i} &= \frac{\alpha_i}{\beta_i}\frac{t_{i-1}}{p_{i-1}} \\
&= (1-\pi_i)^{p'_i - t'_{i-1}} \cdot \frac{t_{i-1}}{p_{i-1}} && \text{defn. } \alpha_i, \beta_i \\
&\leq \exp\left(-\pi_i \cdot (p'_i - t'_{i-1})\right) \cdot \frac{t_{i-1}}{p_{i-1}} && 1 - x \leq e^{-x} \\
&\leq \exp\left(-(1-o(1)) \cdot \frac{1}{K}\left(\frac{p_i}{t_{i-1}} - 1\right)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && \text{defn. } \pi_i,\ \tfrac{p'_i}{t'_{i-1}} = (1-o(1))\tfrac{p_i}{t_{i-1}} \\
&\leq \exp\left(-(1-o(1)) \cdot \frac{1}{K}\left(\frac{\beta_i p_{i-1}}{t_{i-1}} - 1\right)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && \text{defn. } p_i \\
&\leq \exp\left(-(1-o(1)) \cdot \frac{1}{K}\left(e^{-1/K}(1+\epsilon) - 1\right)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && p_{i-1}/t_{i-1} \geq (1+\epsilon) \\
&\leq \exp\left(-(1-o(1)) \cdot \frac{((1-1/K)(1+\epsilon) - 1)}{K}\right) \cdot \frac{t_{i-1}}{p_{i-1}} && e^{-x} \geq 1 - x \\
&= \exp\left(-(1-o(1)) \cdot \frac{\epsilon^2}{4(1+\epsilon)}\right) \cdot \frac{t_{i-1}}{p_{i-1}} && K = 2(1+\epsilon)/\epsilon
\end{aligned}$$

Therefore, the first stage ends after at most $(1+o(1))\frac{4(1+\epsilon)}{\epsilon^2}\ln(1.1Ke^{2/K})$ iterations. Let j be the first iteration when the second stage begins. For $i > j$, we show that $1/\alpha_i$ has an

exponential tower growth.

$$
\begin{aligned}
&\alpha_i = (1-\pi_i)^{p'_i} \\
&\leq \exp\left(-(1-o(1))\frac{1}{K}\cdot\frac{p_i}{t_{i-1}}\right) && 1-x \leq e^{-x} \\
&\leq \exp\left(-(1-o(1))\frac{1}{K}\cdot\frac{\beta_i p_{i-1}}{t_{i-1}}\right) && \text{defn. } p_i \\
&\leq \exp\left(-(1-o(1))\frac{1}{K}\cdot\frac{\beta_{i-1}}{\alpha_{i-1}}\cdot\frac{\beta_i p_{i-2}}{t_{i-2}}\right) && \frac{p_{i-1}}{t_{i-1}} = \frac{\beta_{i-1}}{\alpha_{i-1}}\frac{p_{i-2}}{t_{i-2}} \\
&\leq \exp\left(-(1-o(1))\frac{1}{K}\cdot\frac{e^{-2/K}}{\alpha_{i-1}}\cdot\frac{p_{i-2}}{t_{i-2}}\right) && \beta_i \geq e^{-1/K} \\
&\leq \exp\left(-1/\alpha_{i-1}\right) && \frac{t_{i-2}}{p_{i-2}} < \frac{1}{1.1Ke^{2/K}}
\end{aligned}
$$

Therefore, $\frac{1}{\alpha_{j+\log^* D+1}} \geq \underbrace{e^{e^{\cdots^e}}}_{\log^* D} \geq D$, and so $t_{j+\log^* D+1} \leq \max(\alpha_{j+\log^* D+1}\cdot D, T) = T$. □

On the other hand, we show the bound on the palette size remains large throughout the algorithm.

Lemma 2.4.5. *$p'_i = D^{1-o(1)}$ for $i = O(\log^* D)$.*

Proof. $p'_i = (1-\delta)^i p_i \geq (1-\delta)^i \prod_{j=1}^{i}\beta_j D \geq (1-\delta)^i e^{-i/K} D = (1-o(1))D^{-\frac{i}{K\log D}}\cdot D = D^{1-o(1)}$. □

In the following we shall show how to ensure that for each iteration i the palette sizes are lower bounded by p'_i and the c-degrees are upper bounded by t'_i. For convenience let $H_i(u)$ denote the event that $|P_i(u)| \geq p'_i$ and $\deg_{i,c}(u) \leq t'_i$ for u and $c \in P_{i-1}(u)$. Let H_i denote the event that $H_i(u)$ holds for every $u \in G_i$.

Lemma 2.4.6. *Suppose that H_{i-1} holds, then* $\Pr(|P_i(u)| < (1-\delta)\beta_i|P_{i-1}(u)|) < e^{-\Omega(\delta^2 p'_i)}$.

Proof. Consider a color $c \in P_{i-1}(u)$. The probability that c remains in $P_i(u)$ is exactly β_i. Since the event that c remains in $P_i(u)$ is independent among other colors, by a Chernoff Bound, $\Pr(|P_i(u)| < (1-\delta)\beta_i|P_{i-1}(u)|) < e^{-\Omega(\delta^2 p_{i-1})}$. □

Lemma 2.4.7. *Suppose that H_{i-1} holds, then* $\Pr(\deg_{i,c}(u) > (1+\delta)\cdot\max(\alpha_i \cdot \deg_{i-1,c}(u), T)) < e^{-\Omega(\delta^2 T)} + D\cdot e^{-\Omega(\delta^2 p'_i)}$.

Proof. Let $x_1, \dots x_k \in N_{i-1,c}(u)$ be the c-neighbors of u, ordered by their ID. Let $\mathcal{E}_j$ denote the event that $|P_i(x_j)| \geq p'_i$, where $\Pr(\overline{\mathcal{E}_j}) < e^{-\Omega(\delta^2 p'_i)}$ by Lemma 2.4.6.

Let X_i denote the event that x_i remains uncolored after iteration i. Let $\vec{X_j}$ denote the shorthand for $(X_1, \dots, X_j)$. We will show that for any realization of $\vec{X}_{j-1}$, $\Pr(X_j \mid \vec{X}_{j-1}, \mathcal{E}_1, \dots, \mathcal{E}_j) \leq \alpha_i$. Then we can apply Lemma A.5, which is a variant of Chernoff bound that works when conditioning on a sequence of likely events.

Let $U_2 = N_{i-1}(N_{i,c}(u)) \setminus N_{i,c}(u)$ be the neighbors of the c-neighbors excluding the c-neighbors themselves ($u \in U_2$ unless $\deg_{i-1,c}(u) = 0$). First, notice that the events $\vec{X}_{j-1}$ and $\mathcal{E}_1 \dots, \mathcal{E}_j$ are functions of $S_i(U_2), S_i(x_1), \dots, S_i(x_{j-1}), K_i(x_1), \dots, K_i(x_j)$. Therefore, we can instead show that under any realization of $S_i(U_2), S_i(x_1), \dots, S_i(x_{j-1}), K_i(x_1), \dots, K_i(x_j)$ subject to the events $\mathcal{E}_1 \dots, \mathcal{E}_j$ hold, $\Pr(X_j \mid S_i(U_2), S_i(x_1), \dots, S_i(x_{j-1}), K_i(x_1), \dots, K_i(x_j)) \leq \alpha_i$.

Obviously for any $c' \in P_{i-1}(x_j)$,

$$\Pr(c' \in S_i(x_j) \mid S_i(U_2), S_i(x_1), \dots, S_i(x_{j-1}), K_i(x_1), \dots, K_i(x_j)) = \pi_i.$$

Therefore,

$$\begin{aligned}
&\Pr(X_j \mid S_i(U_2), S_i(x_1), \dots, S_i(x_{j-1}), K_i(x_1), \dots, K_i(x_j)) \\
&\leq (1 - \Pr(c' \in S_i(x_j) \mid S_i(U_2), S_i(x_1), \dots, S_i(x_{j-1}), K_i(x_1), \dots, K_i(x_j)))^{|P_i(u)|} \\
&\leq (1-\pi_i)^{p'_i} = \alpha_i.
\end{aligned}$$

Therefore, by Lemma A.5 and Corollary A.2, and note the fact that $\sum_j \Pr(\overline{\mathcal{E}}_j) \leq D\cdot e^{-\Omega(\delta^2 p'_i)}$, we have $\Pr(\deg_{i,c}(u) > (1+\delta)\cdot\max(\alpha_i \cdot \deg_{i-1,c}(u), T)) \leq e^{-\Omega(\delta^2 T)} + D\cdot e^{-\Omega(\delta^2 p'_i)}$. □

Corollary 2.4.8. *Suppose that H_{i-1} holds,* $\Pr(\overline{H}_i(u)) \leq D\cdot e^{-\Omega(\delta^2 T)} + 2D^2\cdot e^{-\Omega(\delta^2 p'_i)}$.

Proof. By taking union bound over the event in Lemma 2.4.6 and the events in Lemma 2.4.7 over each $c \in P_{i-1}(u)$, we get the desired result. □

Let r be the first iteration such that $t_r = T$. If H_r holds, then $\deg_{r,c}(u) \le t'_r \le (1+\delta)^r t_r \le (1+o(1))t_r \le 2T$ for all u and c. Now we switch to the following analysis, which shows the algorithm terminates in a constant number of iterations. For $i > r$, we define $t'_i = t'_{i-1} \cdot \frac{T}{p'_i}$. The definition for the rest of parameters remain the same. By Lemma 2.4.5, if D is large enough, we can assume that $p'_i \ge D^{1-0.8\gamma}$ for $i = r + \lceil 1/(0.1\gamma) \rceil$, since $r + \lceil 1/(0.1\gamma) \rceil = O(\log^* D)$. Then from the definition of t'_i, it shrinks to less than one in $\lceil \frac{1}{0.1\gamma} \rceil$ iterations, since $T/p'_i \le D^{-0.1\gamma}$ and $t'_{r+1/(0.1\gamma)} < (D^{-0.1\gamma})^{\lceil 1/(0.1\gamma) \rceil} \cdot t'_r < 1$.

Now we will show that under this new defintion of t_i for $i > r$, $H_i(u)$ is likely to hold, provided that H_{i-1} holds.

Lemma 2.4.9. *Suppose that H_{i-1} is true where $i > r$, then* $\Pr(\deg_{i,c}(u) > t'_i) < e^{-\Omega(T)} + D \cdot e^{-\Omega(\delta^2 p'_i)}$

Proof. Let $x_1, \dots x_k \in N_{i-1,c}(u)$ be the c-neighbors of u, ordered by their ID in the increasing order. Let $\mathcal{E}_j$ denote the event that $|P_i(x_j)| \ge p'_i$. Note that $\Pr(\overline{\mathcal{E}}_j) \le e^{-\Omega(\delta^2 p'_i)}$. As we have shown in the proof of Lemma 2.4.7, $\Pr(X_j \mid \vec{X}_j, \mathcal{E}_1, \dots, \mathcal{E}_j) \le \alpha_i$. Therefore,

$$
\begin{aligned}
&\Pr(\deg_{i,c}(u) > t'_i) \\
&= \Pr\left(\deg_{i,c}(u) > \left(\frac{t'_{i-1}}{\alpha_i t'_{i-1}}\right) \cdot \alpha_i t'_{i-1}\right)
\end{aligned}
$$

Applying Lemma A.5 and Corollary A.2 with $1+\delta = t'_i/(\alpha_i t'_{i-1})$, and noticing that $\alpha_i \deg_{i-1,c}(u) \le \alpha_i t'_{i-1}$, the probability above is bounded by

$$
\begin{aligned}
&\le \exp\left(-\alpha_i t'_{i-1}\left(\frac{t'_i}{\alpha_i t'_{i-1}}\ln\frac{t'_i}{\alpha_i t'_{i-1}} - \left(\frac{t'_i}{\alpha_i t'_{i-1}} - 1\right)\right)\right) + De^{-\Omega(\delta^2 p'_i)} \\
&\le \exp\left(-t'_i\left(\ln\frac{t'_i}{\alpha_i t'_{i-1}} - 1\right)\right) + De^{-\Omega(\delta^2 p'_i)} \\
&= \exp\left(-t_i\left(\ln\left(\frac{1}{\alpha_i}\right) - \ln\left(\frac{et'_{i-1}}{t'_i}\right)\right)\right) + De^{-\Omega(\delta^2 p'_i)} \\
&\le \exp\left(-t'_i\left((1-o(1))\frac{p'_i}{Kt'_{i-1}} - \ln\left(\frac{et'_{i-1}}{t'_i}\right)\right)\right) + De^{-\Omega(\delta^2 p'_i)} && \ln\frac{1}{\alpha_i} = (1-o(1))\frac{p'_i}{Kt'_{i-1}} \\
&\le \exp\left(-\left((1-o(1))\frac{T}{K} - t'_i\ln(eD)\right)\right) + De^{-\Omega(\delta^2 p'_i)} && \text{defn. } t'_i \text{ and } t'_{i-1}/t'_i < D \\
&\le \exp\left(-T\left(\frac{(1-o(1))}{K} - \frac{t'_{i-1}}{p'_i}\ln(eD)\right)\right) + De^{-\Omega(\delta^2 p'_i)} \\
&\le \exp\left(-T\left((1-o(1))\frac{1}{K} - \frac{2\ln(eD)}{D^{0.1\gamma}}\right)\right) + De^{-\Omega(\delta^2 p'_i)} && \frac{t'_{i-1}}{p'_i} \le \frac{2T}{p'_i} \le \frac{2}{D^{0.1\gamma}} \\
&\le \exp\left(-\Omega(T)\right) + De^{-\Omega(\delta^2 p'_i)} \square
\end{aligned}
$$

Suppose that H_{i-1} holds, by taking the union bound over all the events $P_i(u) \ge p'_i$ for all $u \in G_{i-1}$ and $\Pr(\deg_{i,c}(u) > t'_i)$ for all $u \in G_{i-1}$ and all $c \in P_{i-1}(u)$, we get that $\Pr(\overline{H}_i(u)) \le D \cdot e^{-\Omega(T)} + 2D^2 \cdot e^{-\Omega(\delta^2 p'_i)}$.

Therefore, we conclude that for each iteration $i \ge 1$, if H_{i-1} holds, then $\Pr(\overline{H}_i(u)) \le D \cdot \exp(-\Omega(\delta^2 T)) + 2D^2 \cdot \exp(-\Omega(\delta^2 p'_i)) \le \exp(-D^{1-0.95\gamma})$ for large enough D. Now we want to ensure that H_i holds for every iteration i. If H_{i-1} is true, then $\Pr(\overline{H}_i(u)) \le \exp\left(-D^{1-0.95\gamma}\right)$. If $D^{1-\gamma} \ge \log n$, then each of the bad event occur with probability at most $1/\operatorname{poly}(n)$. Since there are $O(n)$ events, by the union bound, H_i holds w.h.p. On the other hand, if $D^{1-\gamma} \le \log n$, then we can use the LLL algorithm to make H_i hold w.h.p. The probability of the failure events are bounded by $p = \exp\left(-D^{1-0.95\gamma}\right)$. Each event depends on at most $d = O(\Delta^2)$ other events, since each event only depends on the outcomes of the random variables in its neighborhood. Therefore, $epd^2 \le \exp(-D^{1-\gamma})$ and we can apply the simple LLL algorithm to make all the events hold w.h.p. in $O(\log_{1/epd^2} n) \le O(\log n / D^{1-\gamma})$ iterations.

By Lemma 2.4.4 and the fact that t_i shrinks to 1 in a constant number of iterations after $i > r$, the algorithm uses $O(\log^* D)$ iterations. Each iteration uses $\max(1, O(\log n/D^{1-\gamma}))$ rounds. The total number of rounds is therefore $O(\log^* D \cdot \max(1, O(\log n/D^{1-\gamma})))$.

Chapter 3

Coloring in Triangle-Free Graphs

3.1 Introduction

A proper t-coloring of a graph $G = (V, E)$ is an assignment from V to $\{1, \ldots, t\}$ (colors) such that no edge is monochromatic, or equivalently, each color class is an independent set. The *chromatic number* $\chi(G)$ is the minimum number of colors needed to properly color G. Let Δ be the maximum degree of the graph. It is easy to see that sometimes $\Delta + 1$ colors are necessary, e.g., on an odd cycle or a $(\Delta + 1)$-clique. Brooks' celebrated theorem [17] states that these are the *only* such examples and that every other graph can be Δ-colored. Vizing [170] asked whether Brooks' Theorem can be improved for triangle-free graphs. In the 1970s Borodin and Kostochka [16], Catlin [18], and Lawrence [117] independently proved that $\chi(G) \leq \frac{3}{4}(\Delta + 2)$ for triangle-free G, and Kostochka (see [89]) improved this bound to $\chi(G) \leq \frac{2}{3}(\Delta + 2)$.

Existential Bounds. Better asymptotic bounds were achieved in the 1990s by using an iterated approach, often called the "Rödl Nibble". The idea is to color a very small fraction of the graph in a sequence of rounds, where after each round some property is guaranteed to hold with some small non-zero probability. Kim [104] proved that in any girth-5 graph G, $\chi(G) \leq (1 + o(1))\frac{\Delta}{\ln \Delta}$. This bound is optimal to within a factor-2 under *any* lower bound on girth. (Constructions of Kostochka and Masurova [107] and Bollobás [14] show that there is a graph G of arbitrarily large girth and $\chi(G) > \frac{\Delta}{2 \ln \Delta}$.) Building on [104], Johansson

(see [125]) proved that $\chi(G) = O(\frac{\Delta}{\ln \Delta})$ for any triangle-free (girth-4) graph G.[1] In relatively recent work Jamall [86] proved that the chromatic number of triangle-free graphs is at most $(67 + o(1))\frac{\Delta}{\ln \Delta}$.

Algorithms. Grable and Panconesi [74] gave a distributed algorithm that Δ/k-colors a girth-5 graph in $O(\log n)$ time, where $\Delta > \log^{1+\epsilon} n$ and $k \leq \delta \ln \Delta$ for any $\epsilon > 0$ and some $\delta < 1$ depending on ϵ.[2] Jamall [87] showed a sequential algorithm for $O(\Delta/\ln \Delta)$-coloring a triangle-free graph in $O(n\Delta^2 \ln \Delta)$ time, for any $\epsilon > 0$ and $\Delta > \log^{1+\epsilon} n$.

Note that there are *two* gaps between the existential [86,104,125] and algorithmic results [74, 87]. The algorithmic results use a constant factor more colors than necessary (compared to the existential bounds) and they only work when $\Delta \geq \log^{1+\Omega(1)} n$ is sufficiently large, whereas the existential bounds hold for all Δ.

New Results. We give new distributed algorithms for (Δ/k)-coloring triangle-free graphs that simultaneously improve on both the existential and algorithmic results of [74,86,87,125]. Our algorithms run in $O(\log n)$ time for *all* Δ and in $O(k + \log^* n)$ time for Δ sufficiently large. Moreover, we prove that the chromatic number of triangle-free graphs is $(4+o(1))\frac{\Delta}{\ln \Delta}$.

Theorem 3.1.1. *Fix a constant $\epsilon > 0$. Let Δ be the maximum degree of a triangle-free graph G, assumed to be at least some Δ_ϵ depending on ϵ. Let $k \geq 1$ be a parameter such that $k \leq \frac{1}{4}(1-2\epsilon)\ln \Delta$. Then G can be (Δ/k)-colored, in time $O(k+\log^* \Delta)$ if $\Delta^{1-\frac{4k}{\ln \Delta}-\epsilon} = \Omega(\ln n)$, and, for any Δ, in time on the order of*

$$(k + \log^* \Delta) \cdot \frac{\ln n}{\Delta^{1-\frac{4k}{\ln \Delta}-\epsilon}} \;=\; O(\log n).$$

The first time bound comes from an $O(k + \log^* \Delta)$-round procedure, each round of which succeeds with probability $1 - 1/\operatorname{poly}(n)$. However, as Δ decreases the probability of failure tends to 1. To enforce that each step succeeds with high probability we use our simple

[1] We are not aware of any extant copy of Johansson's manuscript. It is often cited as a DIMACS Technical Report, though no such report exists. Molloy and Reed [125] reproduced a variant of Johansson's proof showing that $\chi(G) \leq 160\frac{\Delta}{\ln \Delta}$ for triangle-free G.

[2] They claimed that their algorithm could also be extended to triangle-free graphs. Jamall [87] pointed out a flaw in their argument.

distributed algorithm for Lovász Local Lemma in Chapter 2 optimized for the parameters of our problem.

Theorem 3.1.1 has a complex tradeoff between the minimum threshold Δ_ϵ, the number of colors, and the threshold for Δ beyond which the running time becomes $O(\log^* n)$. The following corollaries highlight some interesting parameterizations of Theorem 3.1.1.

Corollary 3.1.2. *The chromatic number of triangle-free graphs with maximum degree Δ is at most $(4 + o(1))\Delta/\ln\Delta$.*

Proof. Fix an $\epsilon' > 0$ and choose $k = \ln\Delta/(4+\epsilon')$ and $\epsilon = \epsilon'/(2(4+\epsilon'))$. Theorem 3.1.1 states that for Δ at least some $\Delta_{\epsilon'}$, the chromatic number is at most $(4+\epsilon')\Delta/\ln\Delta$. Now let $\epsilon' = o(1)$ be a function of Δ tending slowly to zero. (The running time of the algorithm that finds such a coloring is never more than $O(\log n)$.) □

Corollary 3.1.3. *Fix any $\delta > 0$. A $(4+\delta)\Delta/\ln\Delta$-coloring of an n-vertex triangle-free graph can be computed in $O(\log^* n)$ time, provided $\Delta > (\ln n)^{(4+\delta)\delta^{-1}+o(1)}$ and n is sufficiently large.*

Proof. Set $k = \ln\Delta/(4+\delta)$ and let $\epsilon = o(1)$ tend slowly to zero as a function of n. If we have

$$\Delta^{1-4k/\ln\Delta-\epsilon} = \Delta^{1-4/(4+\delta)-\epsilon} = \Delta^{\delta(4+\delta)^{-1}-\epsilon} = \Omega(\ln n),$$

or equivalently, $\Delta > (\ln n)^{\delta^{-1}(4+\delta)+o(1)}$, then a $(4+\delta)\Delta/\ln\Delta$-coloring can be computed in $O(\log^* n)$ time. (For n sufficiently large and ϵ tending slowly enough to zero, the lower bound on Δ also implies $\Delta > \Delta_\epsilon$.) □

Theorem 3.1.1 also shows that some colorings can be computed in *sub*logarithmic time, even when Δ is too small to achieve an $O(\log^* n)$ running time.

Corollary 3.1.4. *Fix a $\delta > 0$ and let $k = o(\ln\Delta)$. If $\Delta > (\ln n)^\delta$, a (Δ/k)-coloring can be computed in $(\ln n)^{1-\delta+o(1)}$ time.*

Proof. Let $\epsilon = o(1)$ tend slowly to zero as a function of n. The running time of Theorem 3.1.1

is on the order of

$$(k+\log^* \Delta)\cdot \frac{\ln n}{\Delta^{1-\frac{4k}{\ln \Delta}-\epsilon}} = O\left(\frac{\ln n}{\Delta^{1-o(1)-\epsilon-\ln k/\ln \Delta}}\right)$$
$$= (\ln n)^{1-\delta+o(1)}.\square$$

Our result also extends to girth-5 graphs with $\Delta^{1-\frac{4k}{\ln \Delta}-\epsilon}$ replaced with $\Delta^{1-\frac{k}{\ln \Delta}-\epsilon}$. This change allows us to $(1+o(1))\Delta/\ln\Delta$-color such graphs. Our algorithm can clearly be applied to trees (girth ∞). Elkin [45] noted that with Bollobás's construction [14], Linial's lower bound [118] on coloring trees can be strengthened to show that it is impossible to $o(\Delta/\ln\Delta)$-color a tree in $o(\log_\Delta n)$ time. We prove that it *is* possible to $(1+o(1))\Delta/\ln\Delta$-color a tree in $O(\log\Delta+\log_\Delta\log n)$ time.

Without modifying the analysis, our results extend to list-coloring triangle-free graphs and girth-5 graphs. E.g., we can $(4+o(1))\frac{\Delta}{\ln\Delta}$-list-color triangle-free graphs. However, our result for trees cannot be extended for list-coloring. The algorithm reserves a set of colors for a final coloring phase and these colors must be in the palette of *every* vertex. In list-coloring, it is not possible to reserve such a set of colors.

Technical Overview. Intuitively, consider a vertex u with its Δ neighbors. Suppose that each of its neighbor is colored with a color from one of the $c\Delta/\ln\Delta$ colors uniformly at random, where c is a constant. Then the expected number of colors not chosen by u's neighbor is at least $\Delta\cdot(1-1/(c\Delta/\ln\Delta))^\Delta \sim \Delta^{1-1/c}$. When $c>1$, it is likely there will be colors not colored by u's neighbor and so u can be colored by using one of them. The iterated approaches of [74, 86, 104, 125] manage to achieve the situation where each vertex in the neighborhood is colored uniformly at random, round by round.

In the iterated approaches, each vertex u maintains a *palette*, which consists of the colors that have not been selected by its neighbors. To obtain a t-coloring, each palette consists of colors $\{1,\ldots,t\}$ initially. In each round, each uncolored u tries to assign itself a color (or colors) from its palette, using randomization to resolve the conflicts between itself and the neighbors. The *c-degree* of u is defined to be the number of its neighbors whose palettes contain c. In Kim's algorithm [104] for girth-5 graphs, the properties maintained for each round are that the c-degrees are upper bounded and the palette sizes are lower bounded.

In girth-5 graphs the neighborhoods of the neighbors of u only intersect at u and therefore have a negligible influence on each other, that is, whether c remains in one neighbor's palette has little influence on a different neighbor of u. Due to this independence one can bound the c-degree after an iteration using standard concentration inequalities. In triangle-free graphs, however, there is no guarantee of independence. If two neighbors of u have identical neighborhoods, then after one iteration they will either both keep or both lose c from their palettes. In other words, the c-degree of u is a random variable that may not have any significant concentration around its mean. Rather than bound c-degrees, Johansson [125] bounded the entropy of the remaining palettes so that each color is picked nearly uniformly in each round. Jamall [86] claimed that although each c-degree does not concentrate, the *average* c-degree (over each c in the palette) does concentrate. Moreover, it suffices to consider only those colors within a constant factor of the average in subsequent iterations.

Our (Δ/k)-coloring algorithm performs the same coloring procedure in each round, though the behavior of the algorithm has two qualitatively distinct phases. In the first $O(k)$ rounds the c-degrees, palette sizes, and probability of remaining uncolored vertices are very well behaved. Once the available palette is close to the number of uncolored neighbors, the probability a vertex remains uncolored begins to decrease drastically in each successive round, and after $O(\log^* n)$ rounds all vertices are colored, w.h.p.

Our analysis is similar to that of Jamall [86] in that we focus on bounding the average of the c-degrees. However, our proof needs to take a different approach, for two reasons. First, to obtain an efficient *distributed* algorithm we need to obtain a tighter bound on the probability of failure in the last $O(\log^* n)$ rounds, where the c-degrees shrink faster than a constant factor per round. Second, there is a small flaw in Jamall's application of Azuma's inequality in Lemma 12 in [86], the corresponding Lemma 17 in [87], and the corresponding lemmas in [88]. It is probably possible to correct the flaw, though we manage to circumvent this difficulty altogether. See Section 3.6 for a discussion of this issue.

The second phase presents different challenges. The natural way to bound c-degrees using Chernoff-type inequalities gives error probabilities that are exponential *in the c-degree*, which is fine if it is $\Omega(\log n)$ but becomes too large as the c-degrees are reduced in each coloring round. At a certain threshold we switch to a different analysis (along the lines of Schneider

and Wattenhofer [157]) that allows us to bound c-degrees with high probability in the *palette* size, which, again, is fine if it is $\Omega(\log n)$.

In both phases, if we cannot obtain small error probabilities (via concentration inequalities and a union bound) we revert to a distributed implementation of the Lovász Local Lemma algorithm. We show that for certain parameters the symmetric LLL can be made to run in *sub*logarithmic time. For the extensions to trees and the $(\Delta + 1)$-coloring algorithm for triangle-free graphs, when we cannot obtain small error probabilities, we will ignore those bad vertices where error occured. Using the ideas from [11, 12, 154], we can show the size of each component induced by the bad vertices is at most polylog(n). Each component can then be colored separately in parallel by the deterministic algorithms [7, 141], which now runs faster as the size of each subproblem is smaller.

Organization. Section 6.2 introduces some basic probabilistic tools. Section 3.2 presents the general framework for the analysis. Section 3.3 describes the algorithms and discusses what parameters to plug into the framework. Section 3.4 describes extensions of the algorithm to graphs of girth 5, trees, and the $(\Delta + 1)$-coloring problem for triangle-free graphs.

3.2 The Framework

Every vertex maintains a *palette* that consists of all colors not previously chosen by its neighbors. The coloring is performed in rounds, where each vertex chooses zero or more colors in each round. Let G_i be the graph induced by the uncolored vertices after round i, so $G = G_0$. Let $N_i(u)$ be u's neighbors in G_i and let $P_i(u)$ be its palette after round i. The *c-neighbors* $N_{i,c}(u)$ consist of those $v \in N_i(u)$ with $c \in P_i(v)$. Call $|N_i(u)|$ the *degree* of u and $|N_{i,c}(u)|$ the *c-degree* of u after round i. This notation is extended to sets of vertices in a natural way, e.g., $N_i(N_i(u))$ is the set of neighbors of neighbors of u in G_i.

Algorithm 9 describes the iterative coloring procedure. In each round, each vertex u selects a set $S_i(u)$ of colors by including each $c \in P_{i-1}(u)$ independently with some probability π_i to be determined later. If some $c \in S_i(u)$ is not selected by any neighbor of u then u can safely color itself c. In order to remove dependencies between various random variables (and thereby give us access to the standard concentration bounds from Section 6.2) we exclude

colors from u's palette more aggressively than is necessary. First, we exclude any color *selected* by a neighbor, that is, $S_i(N_{i-1}(u))$ does not appear in $P_i(u)$. The probability that a color c is *not* selected by any neighbor is $(1-\pi_i)^{|N_{i-1,c}(u)|}$. Suppose that this quantity is at least some threshold β_i for all c. We force c to be kept with probability *precisely* β_i by putting c in a keep-set $K_i(u)$ with probability $\beta_i/(1-\pi_i)^{|N_{i-1,c}(u)|}$. The probability that $c \in K_i(u) \backslash S_i(N_{i-1}(u))$ is therefore exactly β_i for each c, assuming $\beta_i/(1-\pi_i)^{|N_{i-1,c}(u)|}$ is a valid probability; if it is not then c is *ignored*. Let $\widehat{P}_i(u)$ be what remains of u's palette. Algorithm 9 has two variants. In Variant B, $P_i(u)$ is exactly $\widehat{P}_i(u)$ whereas in Variant A, $P_i(u)$ is the subset of $\widehat{P}_i(u)$ whose c-degrees are sufficiently low, less than $2t_i$, where t_i is a parameter that will be explained below.

1: Include each $c \in P_{i-1}(u)$ in $S_i(u)$ independently with probability π_i.
2: For each c, calculate $r_c = \beta_i/(1-\pi_i)^{|N_{i-1,c}(u)|}$.
3: If $r_c \leq 1$, include $c \in P_{i-1}(u)$ in $K_i(u)$ independently with probability r_c.
4: **return** $(S_i(u), K_i(u))$.

Algorithm 7: Select(u, π_i, β_i)

1: $i \leftarrow 0$
2: **repeat**
3: $\quad i \leftarrow i+1$
4: $\quad$ **for** each $u \in G_{i-1}$ **do**
5: $\quad\quad (S_i(u), K_i(u)) \leftarrow$ Select(u, π_i, β_i)
6: $\quad\quad$ Set $\widehat{P}_i(u) \leftarrow K_i(u) \setminus S_i(N_{i-1}(u))$
7: $\quad\quad$ **if** $S_i(u) \cap \widehat{P}_i(u) \neq \emptyset$ **then**
8: $\quad\quad\quad$ Color u with any color in $S_i(u) \cap \widehat{P}_i(u)$
9: $\quad\quad$ **end if**
10: $\quad\quad$ (Variant A) $P_i(u) \leftarrow \{c \in \widehat{P}_i(u) \mid |N_{i,c}(u)| \leq 2t_i\}$
11: $\quad\quad$ (Variant B) $P_i(u) \leftarrow \widehat{P}_i(u)$
12: $\quad$ **end for**
13: $\quad G_i \leftarrow G_{i-1} \setminus \{$colored vertices$\}$
14: **until** the termination condition occurs

Algorithm 8: Coloring-Algorithm$(G_0, \{\pi_i\}, \{\beta_i\}, \{t_i\})$

The algorithm is parameterized by the sampling probabilities $\{\pi_i\}$, the ideal c-degrees $\{t_i\}$ and the ideal probability $\{\beta_i\}$ of retaining a color. The $\{\beta_i\}$ define how the ideal palette sizes $\{p_i\}$ degrade. Of course, the *actual* palette sizes and c-degrees after i rounds will drift

from their ideal values, so we will need to reason about approximations of these quantities. We will specify the initial parameters and the terminating conditions when applying both variants in Section 3.3.

3.2.1 Analysis A

Given $\{\pi_i\}$, $p_0 = \Delta/k$, $t_0 = \Delta$, and δ, the parameters for Variant A are derived below.

$$\begin{aligned} \beta_i &= (1-\pi_i)^{2t_{i-1}} & \alpha_i &= (1-\pi_i)^{(1-(1+\delta)^{i-1}/2)p'_i} \\ p_i &= \beta_i p_{i-1} & t_i &= \max(\alpha_i\beta_i t_{i-1}, T) \\ p'_i &= (1-\delta/8)^i p_i & t'_i &= (1+\delta)^i t_i \end{aligned} \tag{3.1}$$

Let us take a brief tour of the parameters. The sampling probability π_i will be inversely proportional to t_{i-1}, the ideal c-degree at the end of round $i-1$. (The exact expression for π_i depends on ϵ.) Since we filter out colors with more than twice the ideal c-degree, the probability that a color is not selected by any neighbor is at least $(1-\pi_i)^{2t_{i-1}} = \beta_i$. Note that since $\pi_i = \Theta(1/t_{i-1})$ we have $\beta_i = \Theta(1)$. Thus, we can force all colors to be retained in the palette with probability precisely β_i, making the ideal palette size $p_i = \beta_i p_{i-1}$. Remember that a c-neighbor stays a c-neighbor if it remains uncolored *and* it does not remove c from its palette. The latter event happens with probability β_i. We use α_i as an upper bound on the probability that a vertex remains uncolored, so the ideal c-degree should be $t_i = \alpha_i\beta_i t_{i-1}$. Notice that a vertex remains uncolored if it did not choose any of the colors remaining in the palette, whose size we will show to be at least $(1-(1+\delta)^{i-1}/2)p'_i$. To account for deviations from the ideal we let p'_i and t'_i be approximate versions of p_i and t_i, defined in terms of a small error control parameter $\delta > 0$. In particular, p'_i and t'_i drift from p_i and t_i by a $(1-\delta/8)$ and a $(1+\delta)$ factor in each round. Furthermore, certain high probability bounds will fail to hold if t_i becomes too small, so we will not let it go below a threshold T.

When the graph has girth 5, the concentration bounds allow us to show that $|P_i(u)| \geq p'_i$ and $|N_{i,c}(u)| \leq t'_i$ with certain probabilities. As pointed out by Jamall [86, 87], $|N_{i,c}(u)|$ does not concentrate in triangle-free graphs. He showed that the average c-degree, $\overline{n}_i(u) = \sum_{c\in P_i(u)} |N_{i,c}(u)|/|P_i(u)|$, concentrates and will be bounded above by t'_i with a certain prob-

ability. Since $\overline{n}_i(u)$ concentrates, it is possible to bound the fraction of colors filtered for having c-degrees larger than $2t_i$ using Markov's inequality.

In the following we formalize this tradeoff between the palette size and the average c-degree. Let $\lambda_i(u) = \min(1, |P_i(u)|/p'_i)$, which can be viewed as the amount that $|P_i(u)|$ drifts below p'_i due to filtering out the colors. Define $\mathcal{H}_i(u)$ to be the event that

$$D_i(u) \leq t'_i,$$
$$\text{where, by definition, } D_i(u) = \lambda_i(u)\overline{n}_i(u) + (1 - \lambda_i(u))2t_i.$$

Define $\mathcal{H}_i$ to be the event that $\mathcal{H}_i(u)$ holds for all $u \in G_i$.[3] Observe that $D_i(u)$ can be interpreted as the average of the c-degrees of $P_i(u)$, including $p'_i - |P_i(u)|$ dummy colors whose c-degrees are exactly $2t_i$. Notice that since $(1 - \lambda_i(u))2t_i \leq D_i(u) \leq t'_i$, we have $1 - \lambda_i(u) \leq t'_i/(2t_i) = (1 + \delta)^i/2$. Therefore,

$$|P_i(u)| \geq (1 - (1 + \delta)^i/2)p'_i \tag{3.2}$$

Recall $P_i(u)$ is the palette consisting of colors c for which $|N_{i,c}(u)| \leq 2t_i$.

In the remainder of this section we prove Theorem 3.2.1, which bounds the probability that $\mathcal{H}_i(u)$ holds conditioned on $\mathcal{H}_{i-1}$.

Theorem 3.2.1. *For any vertex $u \in G_{i-1}$,*

$$\Pr(\mathcal{H}_i(u) \mid \mathcal{H}_{i-1}) = \Pr(D_i(u) \leq t'_i \mid \mathcal{H}_{i-1}) \geq 1 - \Delta e^{-\Omega(\delta^2 T)} - (\Delta^2 + 2)e^{-\Omega(\delta^2 p'_i)}.$$

Note that if $\Pr(\mathcal{H}_i(u) \mid \mathcal{H}_{i-1}) = 1/\operatorname{poly}(n)$, we can conclude, by the union bound, that $\Pr(\mathcal{H}_i \mid \mathcal{H}_{i-1})$ is also $1/\operatorname{poly}(n)$. In general we may need to invoke the Lovász Local Lemma to show $\Pr(\mathcal{H}_i \mid \mathcal{H}_{i-1})$ is nonzero.

3.2.2 Proof of Theorem 3.2.1

Clearly $\mathcal{H}_0$ holds initially. By definition $t'_0 = t_0 = \Delta$ and, for all $u \in G$, we have $\lambda_0(u) = 1$ and $D_0(u) \leq \Delta$. Thus, $D_0(u) \leq t'_0$, i.e., $\mathcal{H}_0(u)$ holds for all u. Let i be the current iteration.

[3] This is equivalent to the induction hypothesis of Jamall [86].

We will assume throughout this section that $\mathcal{H}_{i-1}$ holds, that is, all probabilities obtained are implicitly conditioned on $\mathcal{H}_{i-1}$. Remember that the transition of the palette at round i is from $P_{i-1}(u)$ via $\widehat{P}_i(u)$ to $P_i(u)$, where $\widehat{P}_i(u) = K_i(u) \setminus S_i(N_{i-1}(u))$ is the palette before colors c with c-degree larger than $2t_i$ are filtered. Define $\widehat{n}_i(u) = \sum_{c \in \widehat{P}_i(u)} |N_{i,c}(u)|/|\widehat{P}_i(u)|$ to be the average c-degree over the palette $\widehat{P}_i(u)$. If the following two events hold

- $\mathcal{E}_1(u)$: $|\widehat{P}_i(u)| \geq (1-\delta/8)\beta_i |P_{i-1}(u)|$
- $\mathcal{E}_2(u)$: $\widehat{n}_i(u) \leq \alpha_i \beta_i \overline{n}_{i-1}(u) + \delta(1+\delta)^{i-1} t_i$

then $\mathcal{H}_i(u)$ holds as well, as we now argue.

Observe that if $\mathcal{E}_1(u)$ is true, then the ratio $\widehat{\lambda}_i(u) = \min(1, |\widehat{P}_i(u)|/p'_i)$ is at least as large as $\lambda_i(u)$, since by $\mathcal{E}_1(u)$,

$$\frac{|\widehat{P}_i(u)|}{p'_i} \geq \frac{(1-\delta/8)\beta_i |P_{i-1}(u)|}{(1-\delta/8)\beta_i p'_{i-1}} = \frac{|P_{i-1}(u)|}{p'_{i-1}}.$$

Therefore,

$$\widehat{\lambda}_i(u) \geq \lambda_i(u) \geq \lambda_{i-1}(u). \tag{3.3}$$

Consider $\widehat{D}_i(u) = \widehat{\lambda}_i(u)\widehat{n}_i(u) + (1-\widehat{\lambda}_i(u))2t_i$. Compared to $\widehat{D}_i(u)$, $D_i(u)$ can be viewed as the average c-degree of the palette obtained by changing those colors in $\widehat{P}_i(u)$ whose c-degrees are greater than $2t_i$ to dummy colors with c-degrees exactly $2t_i$. Since the average only goes down in this process,

$$D_i(u) \leq \widehat{D}_i(u). \tag{3.4}$$

Notice that $\overline{n}_{i-1}(u) \leq 2t_i$ and that $\mathcal{H}_{i-1}$ implies $\overline{n}_{i-1}(u) \leq D_{i-1}(u) \leq t'_{i-1}$. We will choose $\delta = o(1)$ sufficiently small so that $(1+\delta)^i = 1 + o(1)$ for any iteration index i encountered in the algorithm. Therefore,

$$\begin{aligned}
\widehat{n}_i(u) &\leq \alpha_i \beta_i \overline{n}_{i-1}(u) + \delta(1+\delta)^{i-1} t_i && \text{by } \mathcal{E}_2(u) \\
&\leq \alpha_i \beta_i t'_{i-1} + \delta(1+\delta)^{i-1} t_i \\
&\leq (1+\delta)^{i-1} t_i + \delta(1+\delta)^{i-1} t_i && \alpha_i \beta_i t_{i-1} \leq t_i \\
&= t'_i \leq 2t_i && (1+\delta)^i = 1 + o(1) < 2
\end{aligned} \tag{3.5}$$

Now we have

$$\begin{aligned}
D_i(u) &\le \widehat{D}_i(u) && \text{by (3.4)}\\
&= \widehat{\lambda}_i(u)\widehat{n}_i(u) + (1-\widehat{\lambda}_i(u))2t_i && \text{defn. of } \widehat{D}_i(u)\\
&\le \lambda_{i-1}(u)\widehat{n}_i(u) + (1-\lambda_{i-1}(u))2t_i && \text{by (3.3) and (3.5)}\\
&\le \lambda_{i-1}(u)(\alpha_i\beta_i\overline{n}_{i-1}(u) + \delta(1+\delta)^{i-1}t_i) + (1-\lambda_{i-1}(u))2t_i && \text{by } \mathcal{E}_2(u)\\
&\le \alpha_i\beta_i(\overline{n}_{i-1}(u) + (1-\lambda_{i-1}(u))2t_{i-1}) + \delta(1+\delta)^{i-1}t_i && t_i = \alpha_i\beta_i t_{i-1}\\
&\le \alpha_i\beta_i D_{i-1}(u) + \delta(1+\delta)^{i-1}t_i && \text{defn. of } D_{i-1}(u)\\
&\le \alpha_i\beta_i t'_{i-1} + \delta(1+\delta)^{i-1}t_i && \mathcal{H}_{i-1} : D_{i-1}(u) \le t'_{i-1}\\
&\le (1+\delta)^{i-1}t_i + \delta(1+\delta)^{i-1}t_i && \alpha_i\beta_i t_{i-1} \le t_i\\
&= t'_i && \text{defn. of } t'_i
\end{aligned}$$

It remains to prove that $\mathcal{E}_1(u)$ and $\mathcal{E}_2(u)$ hold with sufficiently high probability.

3.2.3 Analysis of $\mathcal{E}_1(u)$ and $\mathcal{E}_2(u)$

In this section we show that if $\mathcal{H}_{i-1}$ holds (that is, $D_{i-1}(x) \le t'_{i-1}$ for all x), then events $\mathcal{E}_1(u)$ and $\mathcal{E}_2(u)$ only fail with probability exponentially small in p'_i and T.

The step $\widehat{P}_i(u) \leftarrow K_i(u) \setminus S_i(N_{i-1}(u))$ makes each color remain in $\widehat{P}_i(u)$ with probability exactly β_i independently, therefore $\mathrm{E}[|\widehat{P}_i(u)|] = \beta_i|P_{i-1}(u)|$. By Chernoff Bound, we immediately get that $\mathcal{E}_1(u)$ holds with the following probability:

Lemma 3.2.2. $\Pr(\mathcal{E}_1(u)) = \Pr\left(|\widehat{P}_i(u)| \ge (1-\delta/8)\beta_i|P_{i-1}(u)|\right) \ge 1 - e^{-\Omega(\delta^2 p'_i)}$.

The next step is to bound the probability of $\mathcal{E}_2(u)$. Jamall [86–88] attempted to bound $\widehat{n}_i(u)$ by arguing that, for each c, the value of each $|N_{i,c}(u)|$ is independent of $|N_{i,c'}(u)|$ for $c' \ne c$. Thus, the sum $\sum_{c\in\widehat{P}_{i-1}(u)} |N_{i,c}(u)|$ will concentrate. However, they are not independent since a vertex $x \in N_{i-1}(u)$ can affect $|N_{i,c}(u)|$ for all $c \in P_{i-1}(x)$ if x becomes colored in round i.

To fix this, our idea is to break the analysis into two steps. Define the auxiliary c-neighbor set $\widehat{N}_{i,c}(u) = \{x : x \in N_{i-1,c}(u) \text{ and } c \in \widehat{P}_i(x)\}$ to be the set of neighbors $x \in N_{i-1,c}(u)$ with c remaining in $\widehat{P}_i(x)$ regardless of whether x is colored during round i or not.

For the first step, we will show that due to the independence among $|\widehat{N}_{i,c}(u)|$, for each $c \in P_{i-1}(u)$, $\sum_{c\in\widehat{P}_i(u)} |\widehat{N}_{i,c}(u)|$ will concentrate below $\beta_i^2\overline{n}_{i-1}(u)|P_{i-1}(u)|$. For the second step, we will calculate the probability of $|N_{i,c}(u)| \le \alpha_i|\widehat{N}_{i,c}(u)|$ for each $c \in P_{i-1}(u)$ individually.

Finally, by taking the union bound for the first step and the second step for all $c \in P_{i-1}(u)$, we can prove that $\sum_{c\in\widehat{P}_i(u)} |N_{i,c}(u)|$ concentrates below $\alpha_i\beta_i^2\overline{n}_{i-1}(u)|P_{i-1}(u)|$, which is about $\alpha_i\beta_i\overline{n}_{i-1}(u)|\widehat{P}_i(u)|$ by Lemma 3.2.2.

Lemma 3.2.3. $\Pr\left(\sum_{c\in\widehat{P}_i(u)} |\widehat{N}_{i,c}(u)| \le \beta_i^2|P_{i-1}(u)|\left(\overline{n}_{i-1}(u) + \frac{\delta}{4}t_{i-1}\right)\right) \ge 1 - e^{-\Omega(\delta^2 p'_i)}$.

Proof. Let $Y_c = |\widehat{N}_{i,c}(u)|$ if $c \in \widehat{P}_i(u)$, and $Y_c = 0$ otherwise. Observe that since G is triangle-free, two adjacent vertices u and x have disjoint neighborhoods. Also, whether $c \in \widehat{P}_i(u)$ only depends on the colors selected by its neighbors, not itself. Therefore, $\Pr(c \in \widehat{P}_i(x)|c \in \widehat{P}_i(u)) = \beta_i$ for all $c \in P_{i-1}(x)$. By linearity of expectation, $\mathrm{E}[Y_c] = \Pr(c \in \widehat{P}_i(u)) \sum_{x\in N_{i-1,c}(u)} \Pr(c \in \widehat{P}_i(x)|c \in \widehat{P}_i(u)) = \beta_i^2|N_{i-1,c}(u)|$.

It is clear that $\sum_{c\in\widehat{P}_i(u)} |\widehat{N}_{i,c}(u)| = \sum_{c\in P_{i-1}(u)} Y_c$. By linearity of expectation again, we get that $\mathrm{E}\left[\sum_{c\in\widehat{P}_i(u)}|\widehat{N}_{i,c}(u)|\right] = \mathrm{E}\left[\sum_{c\in P_{i-1}(u)} Y_c\right] = \beta_i^2\overline{n}_{i-1}(u)|P_{i-1}(u)|$.

Since each Y_c ranges from 0 to $2t_{i-1}$ and the $\{Y_c\}$ are independent, by Hoeffding's inequality we have

$$
\begin{aligned}
&\Pr\left(\sum_{c\in\widehat{P}_i(u)} |\widehat{N}_{i,c}(u)| \ge \beta_i^2|P_{i-1}(u)|\left(\overline{n}_{i-1}(u) + \frac{\delta}{4}t_{i-1}\right)\right)\\
&= \Pr\left(\sum_{c\in\widehat{P}_i(u)} |\widehat{N}_{i,c}(u)| \ge \mathrm{E}\left[\sum_{c\in\widehat{P}_i(u)} |\widehat{N}_{i,c}(u)|\right] + \frac{\delta}{4}\beta_i^2|P_{i-1}(u)|t_{i-1}\right)\\
&\le \exp\left(-\frac{\delta^2\beta_i^4 t_{i-1}^2|P_{i-1}(u)|^2}{8\sum_{c\in P_{i-1}(u)}(2t_{i-1})^2}\right)\\
&\le \exp\left(-\frac{\delta^2\beta_i^4|P_{i-1}(u)|}{32}\right)\\
&\le \exp\left(-\frac{\delta^2\beta_i^4(1-(1+\delta)^{i-1}/2)p'_{i-1}}{32}\right) \le \exp\left(-\Omega(\delta^2 p'_i)\right) \qquad \text{by (3.2) and note } \beta_i = \Omega(1)
\end{aligned}
$$

□

Next, we are going to bound the number of uncolored neighbors in $\widehat{N}_{i,c}(u)$ for each $c \in P_{i-1}(u)$. Note that we are not conditioning on whether $c \in \widehat{P}_i(u)$ at this point. Instead, we will take the union bound over all $c \in P_{i-1}(u)$ in the end so that the next lemma holds for all $c \in P_{i-1}(u)$.

Lemma 3.2.4. *Fix an iteration i, vertex u, and color $c \in P_{i-1}(u)$. Letting $M = \max\left(\alpha_i|\widehat{N}_{i,c}(u)|, T\right)$, then we have*

$$\Pr\left(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M\right) \geq 1 - e^{-\Omega(\delta^2 T)} - \Delta e^{-\Omega(\delta^2 p'_i)}.$$

Proof. Let $\mathcal{E}$ be the event that $\mathcal{E}_1(x)$ holds for all $x \in N_{i-1,c}(u)$. By Lemma 3.2.2 and the union bound over each $x \in N_{i-1,c}(u)$, $\Pr(\overline{\mathcal{E}}) \leq |N_{i-1,c}(u)|e^{-\Omega(\delta^2 p'_i)} \leq \Delta e^{-\Omega(\delta^2 p'_i)}$. When $\mathcal{E}$ occurs, for all $x \in N_{i-1,c}(u)$, we have:

$$\begin{aligned}
|\widehat{P}_i(x)| &\geq (1-\delta/8)\beta_i|P_{i-1}(x)| && \text{by } \mathcal{E}_1(x)\\
&\geq (1-\delta/8)\beta_i(1-(1+\delta)^{i-1}/2)p'_{i-1} && \text{by (3.2)}\\
&= (1-(1+\delta)^{i-1}/2)p'_i && \text{By defn., } p'_i = (1-\delta/8)\beta_i p'_{i-1}
\end{aligned}$$

Note that the event $\mathcal{E}$ is determined only by the following random variables:

- $K_i(x)$, for all $x \in N_{i-1,c}(u)$, and
- $S_i(w)$, for all $w \in N_{i-1}(N_{i-1,c}(u))$.

Therefore, we can let $\mathcal{E} = \bigcup_\omega \mathcal{E}_\omega$, where the $\{\mathcal{E}_\omega\}$ represent all the possible outcomes of these random variables that imply $\mathcal{E}$. Then, $\Pr(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M \mid \mathcal{E})$ is exactly

$$\sum_\omega \Pr\left(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M \mid \mathcal{E}_\omega\right) \cdot \Pr\left(\mathcal{E}_\omega \mid \mathcal{E}\right)$$

Since $\omega \neq \omega'$ implies $\mathcal{E}_\omega \cap \mathcal{E}_{\omega'} = \emptyset$, $\sum_\omega \Pr(\mathcal{E}_\omega \mid \mathcal{E}) = 1$. It is sufficient to bound $\Pr(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M \mid \mathcal{E}_\omega)$ for each $\mathcal{E}_\omega$. When conditioning on $\mathcal{E}_\omega$, the neighbor set $\widehat{N}_{i,c}(u)$ is determined and the palette $\widehat{P}_i(x)$ for each $x \in \widehat{N}_{i,c}(u)$ is also determined. Furthermore, since G is triangle-free, $N_{i-1}(N_{i-1,c}(u))$ must be disjoint from $\widehat{N}_{i,c}(u)$. This implies that

conditioning on $\mathcal{E}_\omega$ does not have any influence on $S_i(x)$ for $x \in \widehat{N}_{i,c}(u)$. For all $x \in \widehat{N}_{i,c}(u)$, each $c \in \widehat{P}_i(x)$ is selected with probability π_i independently.

Therefore, the probability x remains uncolored conditioned on $\mathcal{E}_\omega$, $\Pr(x \in N_{i,c}(u) \mid \mathcal{E}_\omega)$, must be independent of all other nodes in $\widehat{N}_{i,c}(u)$. Since x is uncolored iff x did not select any color in $\widehat{P}_i(x)$,

$$\Pr\left(x \in N_{i,c}(u) \mid \mathcal{E}_\omega\right) \;\leq\; (1-\pi_i)^{|\widehat{P}_i(x)|} \;\leq\; (1-\pi_i)^{(1-(1+\delta)^{i-1}/2)p'_i} \;=\; \alpha_i.$$

Therefore, $\mathrm{E}\left[N_{i,c}(u) \mid \mathcal{E}_\omega\right] \;\leq\; \alpha_i \left|\widehat{N}_{i,c}(u)\right|$. By applying Corollary A.1 with $M = \max\left(\alpha_i|\widehat{N}_{i,c}(u)|, T\right)$ we get

$$\Pr\left(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M \mid \mathcal{E}_\omega\right) \;\geq\; 1 - e^{-\Omega(\delta^2 T)}$$

and, therefore,

$$\Pr\left(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M \mid \mathcal{E}\right) \;\geq\; 1 - e^{-\Omega(\delta^2 T)}.$$

Since $\Pr(\overline{\mathcal{E}}) \leq \Delta e^{-\Omega(\delta^2 p'_i)}$, by Lemma A.2, we can conclude that

$$\Pr\left(|N_{i,c}(u)| \leq \alpha_i|\widehat{N}_{i,c}(u)| + (\delta/5)M\right) \;\geq\; 1 - e^{-\Omega(\delta^2 T)} - \Delta e^{-\Omega(\delta^2 p'_i)}.$$

□

For convenience we restate Theorem 3.2.1 before proving it. Recall from Section 3.2.1 that $\mathcal{H}_i = \bigcap_{u \in G_i} \mathcal{H}_i(u)$ and $\mathcal{H}_i(u)$ is the event that $D_i(u) \leq t'_i$.

Theorem 3.2.1. For any vertex $u \in G_{i-1}$,

$$\Pr(\mathcal{H}_i(u) \mid \mathcal{H}_{i-1}) \;=\; \Pr(D_i(u) \leq t'_i \mid \mathcal{H}_{i-1}) \;\geq\; 1 - \Delta e^{-\Omega(\delta^2 T)} - (\Delta^2+2)e^{-\Omega(\delta^2 p'_i)}.$$

Proof. By Lemma 3.2.2, Lemma 3.2.3, Lemma 3.2.4, and the union bound, the following

hold with probability at least $1 - \Delta e^{-\Omega(\delta^2 T)} - (\Delta^2 + 2)e^{-\Omega(\delta^2 p'_i)}$.

$$|\widehat{P}_i(u)| \geq (1 - \delta/8)\beta_i |P_{i-1}(u)| \tag{3.6}$$

$$\sum_{c \in \widehat{P}_i(u)} \left|\widehat{N}_{i,c}(u)\right| \leq \beta_i^2 \left|P_{i-1}(u)\right| \left(\overline{n}_{i-1}(u) + \frac{\delta}{4} t_{i-1}\right) \tag{3.7}$$

$$\text{For all } c \in P_{i-1}(u), |N_{i,c}(u)| \leq \alpha_i |\widehat{N}_{i,c}(u)| + (\delta/5) \max\left(\alpha_i \widehat{N}_{i,c}(u), T\right) \tag{3.8}$$

Therefore,

$$\begin{aligned}
&\sum_{c \in \widehat{P}_i(u)} |N_{i,c}(u)| \\
&\leq \sum_{c \in \widehat{P}_i(u)} \left(\alpha_i |\widehat{N}_{i,c}(u)| + (\delta/5) \max\left(\alpha_i |\widehat{N}_{i,c}(u)|, T\right)\right) && \text{by (3.8)} \\
&\leq (1 + \delta/5)\alpha_i \left(\sum_{c \in \widehat{P}_i(u)} |N_{i,c}(u)|\right) + (\delta/5)|\widehat{P}_i(u)|T && \max(a,b) \leq a + b \\
&\leq \beta_i |P_{i-1}(u)|(1 + \delta/5)\left(\alpha_i \beta_i \overline{n}_{i-1}(u) + \frac{\delta}{4}\alpha_i \beta_i t_{i-1}\right) + (\delta/5)|\widehat{P}_i(u)|T && \text{by (3.7)} \\
&\leq \frac{1 + \delta/5}{1 - \delta/8} |\widehat{P}_i(u)| \left(\alpha_i \beta_i \overline{n}_{i-1}(u) + \frac{\delta}{4}\alpha_i \beta_i t_{i-1}\right) + (\delta/5)|\widehat{P}_i(u)|T && \text{by (3.6)}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\widehat{n}_i(u) &= \sum_{c \in \widehat{P}_i(u)} |N_{i,c}(u)| / |\widehat{P}_i(u)| \\
&\leq \frac{1 + \delta/5}{1 - \delta/8}(\alpha_i \beta_i \overline{n}_{i-1}(u) + (\delta/4)\alpha_i \beta_i t_{i-1}) + (\delta/5)T \\
&\leq (1 + \delta/2)(\alpha_i \beta_i \overline{n}_{i-1}(u) + (\delta/4)\alpha_i \beta_i t'_{i-1}) + (\delta/5)T && \text{when } \delta < 1 \\
&\leq (1 + \delta/2)(\alpha_i \beta_i \overline{n}_{i-1}(u) + (\delta/4)\alpha_i \beta_i t'_{i-1}) + (\delta/5)t_i && T \leq t_i \\
&\leq \alpha_i \beta_i \overline{n}_{i-1}(u) + (\delta/2 + \delta/4 + \delta^2/8)\alpha_i \beta_i t'_{i-1} + (\delta/5)t_i && \overline{n}_{i-1}(u) \leq t'_{i-1} \\
&\leq \alpha_i \beta_i \overline{n}_{i-1}(u) + (\delta/2 + \delta/4 + \delta^2/8 + \delta/5)(1 + \delta)^{i-1} t_i && \alpha_i \beta_i t'_{i-1} = t_i(1 + \delta)^{i-1} \\
&\leq \alpha_i \beta_i \overline{n}_{i-1}(u) + \delta(1 + \delta)^{i-1} t_i && \text{when } \delta \leq 2/5
\end{aligned}$$

As we showed in Section 3.2.2, whenever $\mathcal{E}_1(u)$ and $\mathcal{E}_2(u)$ hold, $\mathcal{H}_i(u)$ holds as well. □

3.2.4 Analysis B

Analysis A has a limitation for smaller c-degrees, since the probability guarantee becomes smaller as t_i goes down. Therefore, Analysis A only works well for $t_i \geq T$, where T is a threshold for certain probability guarantees. For example, if we want Theorem 3.2.1 to hold with high probability in n, then we must have $T \gg \log n$.

To get a good probability guarantee below T, we circumvent Chernoff Bound and calculate the probability explicitly. Also, the reduction in the c-degrees we aimed to show is slower than that in Analysis A. In particular, similar to Theorem 12 in [157], the ideal c-degrees decrease by a factor proportional to the ratio between the initial upper bound on the c-degrees and the current palette size.

The parameters for Variant B are chosen based on an initial lower bound on the palette size p_0, upper bound on the c-degree t_0, and error control parameter δ. The selection probability is chosen to be $\pi_i = 1/(t_{i-1}+1)$ and the probability a color remains in a palette $\beta_i = (1-\pi_i)^{t_{i-1}}$. The ideal palette size and its relaxation are $p_i = \beta_i p_{i-1}$ and $p'_i = (1-\delta)^i p_i$. The ideal c-degree is $t_i = \max(\alpha_i t_{i-1}, 1)$, where $\alpha_i = 5t_0/p'_i$.

Define $\mathcal{F}_i(u)$ to be the event that

$$|P_i(u)| \geq p'_i \text{ and, for all } c \in P_i(u),\ |N_{i,c}(u)| < t_i.$$

Let $\mathcal{F}_i$ be the event that $\mathcal{F}_i(u)$ holds for all $u \in G_i$. When analyzing probabilities in iteration i we always condition on $\mathcal{F}_{i-1}$ holding. Although a vertex could lose its c-neighbor if the c-neighbor becomes colored or loses c in its palette, in this analysis, we only use the former to bound its c-degree. Moreover, if $\mathcal{F}_{i-1}(u)$ holds then $\Pr(c \notin S_i(N_{i-1}(u))) > \beta_i$ for all $c \in P_{i-1}(u)$. Thus in Select(u, π_i, β_i), we will not ignore any colors in the palette. Each color remains in the palette with probability exactly β_i. We will write $P_i(u)$ instead of $\widehat{P}_i(u)$ in this section, since they are the same in Variant B.

Theorem 3.2.5. *For any vertex $u \in G_{i-1}$,*

$$\Pr\left(\mathcal{F}_i(u) \mid \mathcal{F}_{i-1}\right) \geq 1 - \Delta e^{-\Omega(t_0)} - (\Delta^2+1)e^{-\Omega(\delta^2 p'_i)}.$$

Proof. By the Chernoff bound,

$$\begin{aligned}\Pr\left(|P_i(u)| \geq p_i'\right) &\geq \Pr\left(|P_i(u)| \geq (1-\delta/8)\beta_i|P_{i-1}(u)|\right)\\ &\geq 1 - e^{-\Omega(\delta^2 p_i')}.\end{aligned}$$

Now fix a $c \in P_{i-1}(u)$. We will derive a bound on the probability that $|N_{i,c}(u)| < t_i$. Similar to the proof of Lemma 3.2.4, define $\mathcal{E}$ to be the event that

$$\text{For all } x \in N_{i-1,c}(u),\ \ |P_i(x)| \geq p_i'.$$

By taking the union bound over all $x \in N_{i-1,c}(u)$, $\Pr(\mathcal{E}) \geq 1 - |N_{i-1,c}(u)|e^{-\Omega(\delta^2 p_i')} \geq 1 - \Delta e^{-\Omega(\delta^2 p_i')}$. The event $\mathcal{E}$ is determined only by the following random variables:

- $K_i(x)$, for all $x \in N_{i-1,c}(u)$ and
- $S_i(w)$, for all $w \in N_{i-1}(N_{i-1,c}(u))$.

Let $\mathcal{E} = \bigcup_\omega \mathcal{E}_\omega$, where the $\{\mathcal{E}_\omega\}$ represents all the possible outcomes of these random variables that imply $\mathcal{E}$. Then, $\Pr(|N_{i,c}(u)| < t_i \mid \mathcal{E})$ is exactly

$$\sum_\omega \Pr(|N_{i,c}(u)| < t_i \mid \mathcal{E}_\omega) \cdot \Pr(\mathcal{E}_\omega \mid \mathcal{E})$$

Since $\sum_\omega \Pr(\mathcal{E}_\omega \mid \mathcal{E}) = 1$, it is sufficient to bound $\Pr(|N_{i,c}(u)| < t_i \mid \mathcal{E}_\omega)$ for each $\mathcal{E}_\omega$. When conditioning on $\mathcal{E}_\omega$, the palette $P_i(x)$ for each $x \in N_{i-1,c}(u)$ is determined. Furthermore, since G is triangle-free, $N_{i-1}(N_{i-1,c}(u))$ must be disjoint from $N_{i-1,c}(u)$. This implies conditioning on $\mathcal{E}_\omega$ does not have any influence on $S_i(x)$ for all $x \in N_{i-1,c}(u)$. For all $x \in N_{i-1,c}(u)$, each $c \in P_i(x)$ is selected with probability π_i independently at round i.

Note that $x \in N_{i-1,c}(u)$ remains uncolored iff no $c \in P_i(x)$ is selected during round i. Therefore,

$$\Pr(x \in N_{i,c}(u) \mid \mathcal{E}_\omega) \leq (1-\pi_i)^{|\widehat{P}_i(x)|} \leq \left(1 - \frac{1}{t_{i-1}+1}\right)^{p_i'}.$$

By the union bound,

$$\begin{aligned}
\Pr\left(|N_{i,c}(u)| \geq t_i \mid \mathcal{E}_\omega\right) &\leq \sum_{\substack{S \subseteq N_{i-1,c}(u) \\ \text{s.t. } |S|=t_i}} \prod_{x \in S} \Pr(x \in N_{i,c}(u) \mid \mathcal{E}_\omega) \\
&\leq \sum_{\substack{S \subseteq N_{i-1,c}(u) \\ \text{s.t. } |S|=t_i}} \left(1 - \frac{1}{t_{i-1}+1}\right)^{p_i' t_i} \\
&< 2^{t_0} \left(1 - \frac{1}{t_{i-1}+1}\right)^{p_i' t_i} && |N_{i-1,c}(u)| \leq t_0 \\
&\leq 2^{t_0} \exp\left(-\frac{p_i' t_i}{t_{i-1}+1}\right) && 1 - x \leq e^{-x} \\
&\leq 2^{t_0} \exp\left(-\frac{p_i' t_i}{2t_{i-1}}\right) && t_{i-1} + 1 \leq 2t_{i-1} \text{ for } t_i \geq 1 \\
&\leq 2^{t_0} \exp\left(-\frac{\alpha_i p_i'}{2}\right) && t_i \geq \alpha_i t_{i-1} \\
&\leq 2^{t_0} \exp\left(-\frac{5}{2} t_0\right) && \text{defn. of } \alpha_i \\
&= \exp\left(-\left(\frac{5}{2} - \ln 2\right) t_0\right)
\end{aligned}$$

Therefore, $\Pr(|N_{i,c}(u)| < t_i \mid \mathcal{E}) \geq 1 - e^{-\Omega(t_0)}$. Since $\Pr(\mathcal{E}) \geq 1 - \Delta e^{-\Omega(\delta^2 p_i')}$ we can conclude, by Lemma A.2, that $\Pr(|N_{i,c}(u)| < t_i) \geq 1 - e^{-\Omega(t_0)} - \Delta e^{-\Omega(\delta^2 p_i')}$. Recall that $\mathcal{F}_i(u)$ states that $|P_i(u)| \geq p_i'$ and $|N_{i,c}(u)| < t_i$, for all $c \in P_{i-1}(u)$. By the union bound, we have

$$\Pr\left(\mathcal{F}_i(u)\right) \geq 1 - \Delta e^{-\Omega(t_0)} - (\Delta^2 + 1) e^{-\Omega(\delta^2 p_i')}.$$

□

3.3 The Coloring Algorithms

Theorem 3.1.1 is established by analyzing a two-phase coloring algorithm: Phase I uses Analysis A and Phase II uses Analysis B. We will first give the parameters for both phases, then present the distributed algorithm that makes the induction hypotheses ($\mathcal{H}_i$ in Theorem

3.2.1 and $\mathcal{F}_i$ in Theorem 3.2.5) hold with high probability in n, for every round i. Notice that we use the terms iteration and round interchangeably.

Let $\epsilon_1 = 1 - \frac{4k}{\ln \Delta} - \frac{2\epsilon}{3}$ and $\epsilon_2 = 1 - \frac{4k}{\ln \Delta} - \frac{\epsilon}{3}$. We will show that upon reaching the terminating condition of Phase I (which will be defined later), we will have $|P_i(u)| \geq \Delta^{\epsilon_2}$ for all $u \in G_i$ and $|N_{i,c}(u)| < \Delta^{\epsilon_1}$ for all $u \in G_i$ and all $c \in P_i(u)$. At this point, for a non-constructive version, we can simply apply the results about list coloring constants [80, 152, 153] to get a proper coloring, since at this point there is an $\omega(1)$ gap between $|N_{i,c}(u)|$ and $|P_i(u)|$ for every $u \in G_i$. One can turn the result of [152] into a distributed algorithm with the aid of our Lovász Local Lemma algorithm to amplify the success probability. However, to obtain an *efficient* distributed algorithm we use Analysis B in Phase II.

Since our result holds for large enough Δ, we can assume whenever necessary that Δ is sufficiently large. The asymptotic notation will be with respect to Δ.

3.3.1 Parameters for Phase I

In this phase, we use Analysis A with the following parameters: $\pi_i = \frac{1}{2Kt_{i-1}+1}$, where $K = 4/\epsilon$ is a constant, $p_0 = \Delta/k$, $t_0 = \Delta$, and $\delta = 1/\log^2 \Delta$. This phase ends after the round when $t_i \leq T \stackrel{\text{def}}{=} \Delta^{\epsilon_1}/3$.

First, we consider the algorithm for at most the first $O(\log \Delta)$ rounds. For these rounds, we can assume the error $(1+\delta)^i \leq \left(1 + \frac{1}{\log^2 \Delta}\right)^{O(\log \Delta)} \leq e^{O(1/\log \Delta)} = 1 + o(1)$ and similarly $(1 - \delta/8)^i \geq \left(1 - \frac{1}{8\log^2 \Delta}\right)^{O(\log \Delta)} \geq e^{-O(1/\log \Delta)} = 1 - o(1)$. We will show the algorithm reaches the terminating condition during these rounds, where the error is under control.

The probability a color is retained, $\beta_i = (1-\pi_i)^{2t_{i-1}} \geq e^{-1/K}$, is bounded below by a constant. The probability a vertex remains uncolored is at most $\alpha_i = (1-\pi_i)^{(1-(1+\delta)^{i-1}/2)p'_i}$.

If we define $C = 1/(4Ke^{1/K})$, then

$$\begin{aligned}
\alpha_i &\le \left(1 - \frac{1}{2Kt_{i-1}+1}\right)^{(1-(1+\delta)^{i-1}/2)p'_i} \\
&\le \exp\left(-\frac{(1-(1+\delta)^{i-1}/2)p'_i}{(2Kt_{i-1}+1)}\right) \\
&\le \exp\left(-\frac{(1-(1+\delta)^{i-1}/2)(1-\delta/8)^i e^{-1/K} p_{i-1}}{(2Kt_{i-1}+1)}\right) && p'_i \ge (1-\delta/8)^i e^{-1/K} p_{i-1} \\
&\le \exp(-(1-o(1))Cp_{i-1}/t_{i-1}) && \text{defn. of } C
\end{aligned}$$

Let $s_i = t_i/p_i$ be the ratio between the ideal c-degree and the ideal palette size. Initially, $s_0 = k$ and $s_i = \alpha_i s_{i-1} \le s_{i-1} e^{-(1-o(1))(C/s_{i-1})}$. Initially, s_i decreases by roughly C in each round until the ratio $s_i \approx C$ is a constant. Then, s_i decreases rapidly in the order of iterated exponentiation. Therefore, it takes $O(k + \log^* \Delta)$ rounds to reach the terminating condition where $t_i \le T$. Our goal is to show upon reaching the terminating condition, the palette size bound p_i is greater than T by some amount, in particular, $p_i \ge 30e^{3/\epsilon}\Delta^{\epsilon_2}$.

Lemma 3.3.1. *Phase I terminates in $(4 + o(1))Ke^{1/K}k + O(\log^* \Delta)$ iterations, where $K = 4/\epsilon$. Moreover, $p_i \ge 30e^{3/\epsilon}\Delta^{\epsilon_2}$ for every iteration i in this phase.*

Proof. Let $s_i = t_i/p_i$ so that $s_0 = k$. Consider the number of rounds in the following stages:

1. $k \ge s_{i-1} \ge \log^* \Delta$: By using the inequality $e^{-x} \le 1 - x + x^2/2$ for $0 \le x \le 1$, we get $s_i \le s_{i-1} e^{-(1-o(1))(C/s_{i-1})} \le s_{i-1} - (1-o(1))(1 - C/(2s_{i-1}))C \le s_{i-1} - (1-o(1))C$ since $s_{i-1} \ge \log^* \Delta$. Therefore, this stage takes $(1 + o(1))(s_0/C)$ rounds.

2. $\log^* \Delta > s_{i-1} \ge C/1.1$: Similarly, $s_i \le s_{i-1} e^{-(1-o(1))(C/S_{i-1})} \le s_{i-1} - (1 - o(1))C/2$, where we assumed $(1 - o(1))C/s_{i-1} \le 1.59$ and applied Lemma A.1, which states that $e^{-x} \le 1 - x/2$ for $0 \le x \le 1.59$. This stage takes $O(\log^* \Delta/C)$ rounds. Notice that the constant 1.1 was arbitrarily chosen from numbers greater than 1 and no more than 1.59.

3. $C/1.1 > s_{i-1}$: At this point $\alpha_{i+1} \le e^{-(1-o(1))C/s_{i-1}} \le e^{-1}$. For any $j \ge i$, $\alpha_j \le e^{-(1-o(1))C/s_{j-1}} \le e^{-(1-o(1))\frac{C}{s_{i-1}\alpha_{j-1}}} \le e^{-1/\alpha_{j-1}}$. Therefore, after $j = \log^* \Delta$ more rounds,

$\alpha_{i+j} \leq 1/\Delta$ and so $t_{i+j} \leq \max(\alpha_{i+j}t_0, T) = \Delta^{\epsilon_1}/3$ terminates Phase I. This stage takes $\log^* \Delta$ rounds.

The total number of rounds is $(1+o(1))(s_0/C)+O(\log^* \Delta) \leq (4+o(1))Ke^{1/K}k+O(\log^* \Delta)$. By the definition of p_i, at the end of Phase I we have:

$$
\begin{aligned}
p_i &= p_0 \prod_{j=1}^{i} \beta_j \\
&\geq \frac{\Delta}{k} e^{-\frac{1}{K}\left((4+o(1))\,Ke^{1/K}k \,+\, O\left(\log^* \Delta\right)\right)} && \beta_j \geq e^{-1/K} \\
&\geq \frac{\Delta}{k}\left(\frac{1}{\Delta}\right)^{\frac{(4+o(1))e^{1/K}k + O(\log^* \Delta)}{\ln \Delta}} \\
&\geq \Delta^{1-\frac{4e^{1/K}k}{\ln \Delta} - o(1)} && k < \ln \Delta \\
&\geq \Delta^{1-\frac{4k}{\ln \Delta} - \frac{1}{K}\frac{4k}{\ln \Delta}\left(1+\frac{1}{K}\right) - o(1)} && \text{by using } e^x \leq 1+x+x^2 \text{ for } |x| \leq 1 \\
&\geq \Delta^{1-\frac{4k}{\ln \Delta} - \frac{\epsilon}{4}(1-2\epsilon)\left(1+\frac{\epsilon}{4}\right) - o(1)} && \text{since } K = 4/\epsilon \text{ and } \tfrac{4k}{\ln \Delta} \leq 1-2\epsilon \\
&\geq \Delta^{1-\frac{4k}{\ln \Delta} - \epsilon/4 - o(1)}
\end{aligned}
$$

□

Thus, for large enough Δ, p_i is at least $30e^{3/\epsilon}\Delta^{\epsilon_2}$, which will be enough for the induction hypothesis to hold with sufficiently high probability. If $\mathcal{H}_i(u)$ holds for every $u \in G_i$ for every round i during this phase, we will have $|P_i(u)| \geq (1-(1+\delta)^i/2)p'_i \geq 10e^{3/\epsilon}\Delta^{\epsilon_2}$ for all $u \in G_i$ and $|N_{i,c}(u)| \leq 2t_i < \Delta^{\epsilon_1}$ for all $u \in G_i$ and all $c \in P_i(u)$ in the end of Phase I.

3.3.2 Parameters for Phase II

In Phase II, we will use Analysis B with the following parameters: $p_0 = 10e^{3/\epsilon}\Delta^{\epsilon_2}$, $t_0 = \Delta^{\epsilon_1}$ and $\delta = 1/\log^2 \Delta$. This phase terminates after $\frac{3}{\epsilon}$ rounds.

First note that the number of rounds $\frac{3}{\epsilon}$ is a constant. We show $p'_i \geq 5\Delta^{\epsilon_2}$ for each round $1 \leq i \leq \frac{3}{\epsilon}$, so there is always a sufficient large gap between the current palette size and the initial c-degree, which implies the shrinking factor of the c-degrees is $\alpha_i = 5t_0/p'_i \leq \Delta^{-\epsilon/3}$.

Since p_i shrinks by at most a $\beta_i \geq e^{-1}$ factor every round, $p'_i \geq (1-\delta)^i p_0 \prod_{j=1}^{i} \beta_j \geq ((1-\delta)e^{-1})^i 10e^{3/\epsilon}\Delta^{\epsilon_2} \geq 5\Delta^{\epsilon_2}$.

Now since $\alpha_i \leq \Delta^{-\epsilon/3}$, after $\frac{3}{\epsilon}$ rounds, $t_i \leq t_0 \prod_{j=1}^{i} \alpha_j \leq \Delta \left(\Delta^{-\epsilon/3}\right)^{\frac{3}{\epsilon}} \leq 1$. The c-degree bound, $t_{\epsilon/3}$, becomes 1. Recall that the induction hypothesis $\mathcal{F}_i(u)$ is the event that $|P_i(u)| \geq p'_i$ and $|N_{i,c}(u)| < t_i$ for all $c \in P_i(u)$. If $\mathcal{F}_i$ holds for every round i in Phase II then, in the end, every uncolored vertex has no c-neighbors, as implied by $|N_{i,c}(u)| < t_i \leq 1$. This means these vertices can be colored with anything remaining in their palettes, which are non-empty.

The leading constant 4 The leading constant 4 stems from filtering out colors whose c-degree exceeds twice of the ideal. In general, if we filter out colors whose c-degree exceeds q times the ideal, then the remaining palette has size at least $(1-1/q)$ of the original one. q affects how fast the ratio t_i/p_i decreases for every round. In particular, it decreases roughly by $1/(q/(1-1/q)Ke^{1/K})$ for every round. Note that the palette size decreases by a fixed rate $\beta_i \sim e^{1/K}$ for each round i and we have to keep it large enough as stated in Lemma 3.3.1 ($p_i \geq 30e^{3/\epsilon}\Delta^{\epsilon_2}$). Given that the number of rounds we allow is fixed, the leading constant we can get depends on how fast the ratio t_i/p_i decreases. Therefore, we choose $q = 2$ to maximize $1/(q/(1-1/q)Ke^{1/K})$, which results in a leading constant of 4.

3.3.3 The Distributed Coloring Algorithm

We will show a distributed algorithm that makes the induction hypothesis in Phase I and Phase II hold with high probability in n.

Fix the round i and assume the inductive hypothesis holds after round $i-1$, which is either $\mathcal{H}_{i-1}$ in Phase I or $\mathcal{F}_{i-1}$ in Phase II. Define $A(u)$ to be the bad event that the induction hypothesis fails at u, that is, $\mathcal{H}_i(u)$ fails in Phase I or $\mathcal{F}_i(u)$ fails in Phase II. Let $p = e^{-\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon}}/(e\Delta^8)$. By Theorem 3.2.1 and 3.2.5 we have

$$\Pr(A(u)) \leq \max\left(\Delta e^{-\Omega\left(\delta^2 T\right)} + (\Delta^2+2)e^{-\Omega\left(\delta^2 p'_i\right)}, \quad \Delta e^{-\Omega(t_0)} + (\Delta^2+1)e^{-\Omega\left(\delta^2 p'_i\right)}\right).$$

Therefore,

$$\begin{aligned}
\Pr(A(u)) &\le \Delta e^{-\Omega\left(\delta^2\Delta^{\epsilon_1}\right)} + (\Delta^2+2)e^{-\Omega\left(\delta^2\Delta^{\epsilon_2}\right)} && T=\Delta^{\epsilon_1}, t_0=\Delta^{\epsilon_1}, p_i' \ge \Delta^{\epsilon_2} \\
&\le \exp\left(-\Omega\left(\delta^2\Delta^{\epsilon_1}\right)+O\left(\log\Delta\right)\right)\Big/(e\Delta^8) && \epsilon_1<\epsilon_2 \\
&\le \exp\left(-\Omega\left(\delta^2\Delta^{\epsilon_1}\right)\right)\Big/(e\Delta^8) && \\
&\le \exp\left(-\Omega\left(\frac{\Delta^{\frac{1}{3}\epsilon}}{\log^4\Delta}\right)\cdot\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon}\right)\Big/(e\Delta^8) && \text{defn. } \epsilon_1 \text{ and } \delta \\
&\le \exp\left(-\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon}\right)\Big/(e\Delta^8) = p && \text{for large enough } \Delta
\end{aligned}$$

If $\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon} > c\log n$, then $p < 1/n^c$. By the union bound over $u \in G_i$, the probability that $\{A(u)\}$ all fail to occur is at least $1-1/n^{c-1}$. In other words, the induction hypothesis ($\mathcal{H}_i$ or $\mathcal{F}_i$) holds after round i with high probability. In this case, $O(k+\log^*\Delta)$ rounds suffice, because each round succeeds with high probability.

On the other hand, if $\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon} < c\log n$ then we apply our parallel resampling algorithm (Algorithm 2) to find a point avoiding all the bad events $\{A(u)\}$, with high probability. The symmetric LLL and its algorithmic versions refer to the following objects and parameters.

- A set $\mathcal{P}$ of random variables over some domain, which may be different for each variable.
- A set $\mathcal{A}$ of "bad" events. Each $A \in \mathcal{A}$ depends only on some subset $\mathrm{vbl}(A) \subseteq \mathcal{P}$ of the variables.
- Define $\Gamma(A) = \{A' \mid A' \neq A \text{ and } \mathrm{vbl}(A') \cap \mathrm{vbl}(A) \neq \emptyset\}$ to be those events that share variables with A. The Γ function induces an undirected dependency graph $G = (\mathcal{A}, \{(A,A') \mid A' \in \Gamma(A)\})$. Let $G_{\mathcal{B}}$ be the subgraph induced by $\mathcal{B} \subseteq \mathcal{A}$.
- Define $d = \max_{A\in\mathcal{A}}|\Gamma(A)|$ and $p = \max_{A\in\mathcal{A}}\Pr(A)$ to be the maximum degree in the dependency graph and the maximum probability of any single bad event.

If $A \in \mathcal{A}$ occurs under an assignment to $\mathcal{P}$ we say it is *violated*. Our LLL algorithm repeatedly selects a set of violated events and *resamples* the variables they depend on, halting when no events are violated. If $|\mathcal{A}| = n$, by Corollary 2.3.4, the running time of our algorithm is $O(\log_{1/epd^2} n)$.

Observe that $A(u)$ depends only on random variables selected by u and vertices at distance 1 or 2 from u. It follows that if $\text{dist}_{G_{i-1}}(u,v) \geq 5$ then $A(u)$ and $A(v)$ are independent. Let $G_{i-1}^{\leq 4}$ be the dependency graph where (u,v) is an edge iff $\text{dist}_{G_{i-1}}(u,v) \leq 4$. The maximum degree in $G_{i-1}^{\leq 4}$ is clearly less than Δ^4.

Therefore, $d < \Delta^4$ and $p = e^{-\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon}}/(e\Delta^8)$. We have $epd^2 \leq e^{-\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon}}$. By Corollary 2.3.4, $O(\log n/\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon})$ resampling rounds will be sufficient. Our algorithm for LLL was described to run on the dependency graph, $G_{i-1}^{\leq 4}$. Though G is the underlying network in our case, we can simulate our algorithm in $G_{i-1}^{\leq 4}$ with constant factor slowdown.

Each of the $O(k + \log^* \Delta)$ rounds consists of $O(\log n/\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon})$ resampling rounds. The total number of rounds is $O\left(k + \log^* \Delta\right) \cdot \left(\log n/\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon}\right)$. Note that this is always at most $O(\log n)$, since $\Delta^{1-\frac{4k}{\ln\Delta}-\epsilon} \geq \Delta^{\epsilon} = \Delta^{\Omega(1)}$. If $\Delta^{1-\frac{4k}{\log\Delta}-\epsilon} = O(\log^{1-\gamma} n)$ for some constant $\gamma > 0$, then the running time is sublogarithmic.

3.4 Extensions

3.4.1 Graphs of Girth at Least 5

For graphs of girth at least 5, existential results [104, 125] show that there exists $(1 + o(1))\Delta/\ln\Delta$-coloring. Grable and Panconesi [74] gave a distributed algorithm that run in $O(\log n)$ time to find a (Δ/k)-coloring for $k = O(\log\Delta)$ when $\Delta \gg \log^{1+\epsilon'} n$ for some constant $\epsilon' > 0$. Since there is a constant hidden in $k = O(\log\Delta)$, the $k = (1+o(1))\Delta/\ln\Delta$-coloring is not obtainable by their algorithm. We close this gap by extending our result for triangle-free graphs and replacing the leading constant 4 by 1.

Theorem 3.4.1. *Fix a constant $\epsilon > 0$. Let Δ be the maximum degree of a girth-5 graph G, assumed to be at least some Δ_ϵ depending on ϵ. Let $k \geq 1$ be a parameter such that $2\epsilon \leq 1 - \frac{k}{\ln\Delta}$. Then G can be (Δ/k)-colored, in time $O(k + \log^* \Delta)$ if $\Delta^{1-\frac{k}{\ln\Delta}-\epsilon} = \Omega(\ln n)$, and, for any Δ, in time on the order of*

$$(k + \log^* \Delta) \cdot \frac{\log n}{\Delta^{1-\frac{k}{\ln\Delta}-\epsilon}} \;=\; O(\log n)$$

In Analysis A, instead of using the inductive hypothesis $\mathcal{H}_i(u)$ and Variant A in Phase I, we shall use Variant B and prove the following induction hypothesis, $\mathcal{Q}_i(u)$:

$$|P_i(u)| \geq p'_i \text{ and, for all } c \in P_i(u),\ |N_{i,c}(u)| \leq t'_i$$

Define $\mathcal{Q}_i$ to be the events that $\mathcal{Q}_i(u)$ holds for all $u \in G_i$. Also, we use definitions with a slightly different error control:

$$\begin{aligned} \beta_i &= (1-\pi_i)^{t'_{i-1}} & \alpha_i &= (1-\pi_i)^{p'_i} \\ p_i &= \beta_i p_{i-1} & t_i &= \max(\alpha_i \beta_i t_{i-1}, T) \\ p'_i &= (1-\delta)^i p_i & t'_i &= \frac{(1+\delta)^i}{\prod_{k=1}^{i}(1-\pi_k)} t_i \end{aligned} \tag{3.9}$$

We use $\pi_i = 1/(1+Kt'_i)$ as the sampling probability in the ith iteration, where $K = 4/\epsilon$. As a consequence β_i is lower bounded by a constant since

$$\beta_i = (1-\pi_i)^{t'_{i-1}} = \left(1 - \frac{1}{1+Kt'_i}\right)^{t'_{i-1}} > \left(1 - \frac{1}{1+Kt'_i}\right)^{t'_i} > e^{-1/K}.$$

Notice that since $\delta = 1/\log^2 \Delta$, for the first $i = O(\log \Delta)$ rounds we have $p'_i = p_i(1-\delta)^i = (1-o(1))p_i$. If we choose $\epsilon_1 = 1 - \frac{k}{\ln \Delta} - \frac{2\epsilon}{3}$ and $\epsilon_2 = 1 - \frac{k}{\ln \Delta} - \frac{\epsilon}{3}$, and end the phase after the first round i when $t_i \leq T \stackrel{\text{def}}{=} \Delta^{\epsilon_1}/3$, then

$$t'_i = \frac{(1+\delta)^i}{\prod_{k=1}^{i}(1-\pi_k)} \cdot t_i \leq \left(\frac{(1+\delta)}{1-(1+K\Delta^{\epsilon_1/3})^{-1}}\right)^i t_i \leq (1+o(1))t_i.$$

Then, Lemma 3.3.1 holds similarly except that the algorithm runs in $(1+o(1))Ke^{1/K}k + O(\log^* \Delta)$ time. Also, one can prove $p_i \geq 30e^{3/\epsilon}\Delta^{\epsilon_2}$ as in the proof of Lemma 3.3.1. The argument for Phase II afterwards will be the same with that in triangle-free graphs.

The remaining task is to bound the failure probability of $\mathcal{Q}_i(u)$, when $\mathcal{Q}_{i-1}$ holds. To show this, notice that it is possible to bound individual c-degrees rather than bounding the average c-degree in graphs of girth at least 5, since the probability a color remains in each neighbor has only a weak correlation. Instead of proving Lemma 3.2.3, we prove the following:

Lemma 3.4.2. *Let* $\beta'_i = \frac{\beta_i}{(1-\pi_i)}$. *If* $\mathcal{Q}_{i-1}$ *holds, then* $\Pr\left(|\widehat{N}_{i,c}(u)| \le (1+\delta/4)\beta'_i t'_{i-1}\right) \ge 1 - e^{-\Omega(\delta^2 T)}$

Proof. Let $x \in N_{i-1,c}(u)$. Whether x loses c in its palette depends on whether x's neighbors chose c. Since G is a graph with girth at least 5, u is the only common neighbor of vertices in $N_{i-1,c}(u)$. The probability that x loses c is almost independent among other vertices in $N_{i-1,c}(u)$. Let I_x be the indicator random variable that $c \in K(x)$ and all of x's neighbors excluding u did not choose c (i.e. $c \in K(x) \setminus S_i(N_{i-1,c}(x) \setminus \{u\})$). Clearly, I_x are independent among all $x \in N_{i-1,c}(u)$ and $\Pr(I_x) = \beta'_i$. Letting $I = \sum_x I_x$, we have $|\widehat{N}_{i,c}(u)| \le I$ and $\mathrm{E}[I] = \beta'_i |N_{i-1,c}(u)|$. Therefore,

$$
\begin{aligned}
&\Pr\left(|\widehat{N}_{i,c}(u)| \le (1+\delta/4)\beta'_i t'_{i-1}\right) && \\
&\ge \Pr\left(I \le (1+\delta/4)\beta'_i t'_{i-1}\right) && |\widehat{N}_{i,c}(u)| \le I \\
&\ge \Pr\left(I \le \mathrm{E}[I] + (\delta/4)\beta'_i t'_{i-1}\right) && \mathrm{E}[I] = \beta'_i |N_{i-1,c}(u)| \le \beta'_i t'_{i-1} \\
&\ge 1 - e^{-\Omega(\delta^2 \beta'_i t'_{i-1})} && \beta'_i t'_{i-1} \ge \mathrm{E}[I] \text{ and by Corollary A.1} \\
&= 1 - e^{-\Omega(\delta^2 T)} && \beta'_i = \Omega(1) \text{ and } t'_{i-1} \ge T
\end{aligned}
$$

□

Combined with Lemma 3.2.4, we get the following:

Corollary 3.4.3. *If* $\mathcal{Q}_{i-1}$ *holds, then* $\Pr\left(\forall c \in P_{i-1}(u),\ |N_{i,c}(u)| \le t'_i\right) \ge 1 - 2\Delta e^{-\Omega(\delta^2 T)} - \Delta^2 e^{-\Omega(\delta^2 p'_i)}$.

Proof. By applying union bound with Lemma 3.4.2 and Lemma 3.2.4 for all $c \in P_{i-1}(u)$, the following holds with probability at least $1 - 2\Delta e^{-\Omega(\delta^2 T)} - \Delta^2 e^{-\Omega(\delta^2 p'_i)}$:

1. $|\widehat{N}_{i,c}(u)| \le (1+\delta/4)\beta'_i t'_{i-1}$
2. $|N_{i,c}(u)| \le \alpha_i |\widehat{N}_{i,c}(u)| + (\delta/5)\max(\alpha_i |\widehat{N}_{i,c}(u)|, T)$

Then, we have:

$$\begin{aligned}
|N_{i,c}(u)| &\le \alpha_i\beta_i'(1+\delta/4)(1+\delta/5)t_{i-1}' + (\delta/5)T && \max(a,b) \le a+b \\
&\le \alpha_i\beta_i'(1+\delta/4)(1+2\delta/5)t_{i-1}' && T \le t_i \le \alpha_i\beta_i t_{i-1} \le \alpha_i\beta_i'(1+\delta/4)t_{i-1} \\
&\le (1+\delta)\alpha_i\beta_i' t_{i-1}' && \delta \le 1 \\
&= t_i' && \text{defn. of } \beta_i' \text{ and } t_i'
\end{aligned}$$

□

Theorem 3.4.4. *For any vertex $u \in G_{i-1}$, $\Pr(\mathcal{Q}_i(u) \mid \mathcal{Q}_{i-1}) \ge 1 - 2\Delta e^{-\Omega(\delta^2 T)} - (\Delta^2+1)e^{-\Omega(\delta^2 p_i')}$.*

Proof. By Chernoff bound, we can get that $\Pr(|P_i(u)| \ge p_i') = \Pr(|P_i(u)| \ge (1-\delta)\beta_i p_{i-1}') \ge \Pr(|P_i(u)| \ge (1-\delta)\beta_i |P_{i-1}(u)|) \ge 1 - e^{-\Omega(\delta^2 p_i')}$. By the union bound and Corollary 3.4.3, we get that $|P_i(u)| \ge p_i'$ and $|N_{i,c}(u)| \le t_{i-1}'$ for all $c \in P_i(u)$ hold with probability at least $1 - 2\Delta e^{-\Omega(\delta^2 T)} - (\Delta^2+1)e^{-\Omega(\delta^2 p_i')}$ □

Since $p_i \ge C_2\Delta^{\epsilon_2}$ and $T = \Delta^{\epsilon_1}/3$, the probability $\mathcal{Q}_i(u)$ fails for u, $2\Delta e^{-\Omega(\delta^2 T)} + (\Delta^2+1)e^{-\Omega(\delta^2 p_i')}$, is bounded by $e^{-\Delta^{1-\frac{k}{\ln\Delta}-\epsilon}}/(e\Delta^8)$ for large enough Δ. As in Section 3.3, depending on how small this probability is, one can either apply the union bounds to get a high success probability or use our resampling algorithm for the Lovász Local Lemma.

3.4.2 Trees

Trees are graphs of infinity girth. According to Theorem 3.4.1, it is possible to get a (Δ/k)-coloring in $O(k + \log^* \Delta)$ time if $\Delta^{1-\frac{k}{\ln\Delta}-\epsilon} = \Omega(\log n)$ and ϵ is a constant less than or equal to $\frac{1}{2}\cdot(1-\frac{k}{\ln\Delta})$. If $\Delta^{1-\frac{k}{\ln\Delta}-\epsilon} = O(\log n)$, we will show that using additional $O(q)$ colors, it is possible to get a $(\Delta/k + O(q))$-coloring in $O\left(k + \log^* n + \frac{\log\log n}{\log q}\right)$ time. In any case, we can find a $(1+o(1))\Delta/\ln\Delta$-coloring in $O(\log\Delta + \log_\Delta \log n)$ rounds by choosing $q = \sqrt{\Delta}$ and $k = \ln\Delta/(1+o(1))$.

The algorithm is the same with the framework of Section 3.4.1, except that at the end of each round we delete the *bad* vertices, which are the vertices that fail to satisfy the

induction hypothesis (i.e. $\mathcal{Q}_i(u)$ in Phase I or $\mathcal{F}_i(u)$ in Phase II). The remaining vertices must satisfy the induction hypothesis. Using the idea from [11, 12, 154], we will show that after $O(k + \log^* \Delta)$ rounds of the algorithm, the size of each component formed by the bad vertices is at most $O(\Delta^4 \log n)$ with high probability.

Barenboim and Elkin's deterministic algorithm [7] obtains an $O(q)$-coloring in $O\left(\frac{\log n}{\log q} + \log^* n\right)$ time for trees (arboricity = 1). We then apply their algorithm on each component formed by bad vertices. Since the size of each component is at most $O(\Delta^4 \log n)$, their algorithm will run in $O\left(\frac{\log\log n + \log \Delta}{\log q} + \log^* n\right)$ time, using the additional $O(q)$ colors. Note that this running time is actually $O\left(\frac{\log\log n}{\log q} + \log^* n\right)$, since $\Delta = O(\log^{1/(1-\frac{k}{\ln \Delta}-\epsilon)} n) = O(\log^{1/\epsilon} n) = \log^{O(1)} n$.

Define $A_i(u)$ be the event the induction hypothesis fails at u in round i ($\mathcal{Q}_i(u)$ fails in Phase I or $\mathcal{F}_i(u)$ fails in Phase II). Since $k < \ln \Delta$, there exists a constant $c_1 > 0$ such that the algorithm always finishes in $c_1 \ln \Delta$ rounds. Let $p = 1/(2c_1 e\Delta^5 \ln \Delta)$. By Theorem 3.2.5 and Theorem 3.4.4 and since $T \geq \Delta^{1-\frac{k}{\ln \Delta}-\frac{2\epsilon}{3}}$ and $p_i' \geq \Delta^{1-\frac{k}{\ln \Delta}-\frac{\epsilon}{3}}$, we have that for large enough Δ,

$$\Pr(A_i(u)) \leq 2\Delta e^{-\Omega(\delta^2 T)} + (\Delta^2 + 1)e^{-\Omega(\delta^2 p_i')} \leq 1/(2c_1 e\Delta^5 \ln \Delta) = p$$

Also note that for $u, v \in G_{i-1}$, $A_i(u)$ and $A_i(v)$ are independent if $\mathrm{dist}_{G_{i-1}}(u, v) \geq 5$, since $A_i(u)$ only depends on variables within distance two from u.

Lemma 3.4.5. *Let $H \subseteq G_{i-1}$ be a connected component with s vertices. There exists a vertex set $V_0 \subseteq H$ such that $|V_0| = \lceil s/\Delta^4 \rceil$ and for any $u, v \in V_0$, $\mathrm{dist}_{G_{i-1}}(u, v) \geq 5$ and $\mathrm{dist}_{G_{i-1}}(u, V_0 \setminus \{u\}) = 5$.*

Proof. Define $B(v, i) = \{u \in H \mid \mathrm{dist}_{G_{i-1}}(v, u) \leq i\}$. Start with an arbitrary vertex $v \in H$. Put v in V_0 and delete $B(v, 4)$ from H. Select a new vertex v' from the remaining vertices in H such that $\mathrm{dist}(v', V_0) = 5$. If the remaining graph is non-empty, then such v' must exist, because H is connected. Repeat this procedure until there are $\lceil s/\Delta^4 \rceil$ vertices in V_0. Since we delete at most Δ^4 vertices in each iteration, the remaining graph will be non-empty until we find $\lceil s/\Delta^4 \rceil$ vertices. □

Suppose that there exists a component H containing s bad vertices in the end of the algorithm. Let $t = \lceil s/\Delta^4 \rceil$, we can extract such a subset $V_0 \subseteq H$ with the property stated in Lemma 3.4.5. We will show that the total possible number of such V_0 will be bounded.

For any V_0, we can map it to a tree with size t in the graph G_{i-1}^5. This is because the vertex set of V_0 is connected in G_{i-1}^5 and we can take any spanning tree of it. The mapping is injective. Therefore, the total number of possilbe V_0 is at most the total possible number of ways to embed an unordered, rooted tree of t vertices in G_{i-1}^5, which is bounded by $ne^t\Delta^{5t}$ [105, p. 397, Exercise 11].

On the other hand, the total possible number schedules for when these t vertices become bad is at most $c_1^t \ln^t \Delta$, since each vertex becomes bad in one of at most $c_1 \ln \Delta$ rounds in our algorithm. For those $u \in V_0$ who become bad at round i, each failure happens with probability at most p independently. Therefore,

$$
\begin{aligned}
&\Pr(\exists H \text{ s.t. } |H| \geq s \text{ and } v \text{ is bad}, \forall v \in H) \\
&\leq \sum_{\substack{\text{tree } T \subseteq G_{i-1}^5 \\ |T| = s/\Delta^4}} \Pr(v \text{ is bad}, \forall v \in T) \\
&\leq \sum_{\substack{\text{tree } T \subseteq G_{i-1}^5 \\ |T| = s/\Delta^4}} \sum_{\substack{B_1, \ldots, B_{c_1 \ln \Delta} \subseteq T \\ \bigcup B_i = T, B_i \cap B_j = \emptyset}} \prod_i \Pr(B_i \text{ become bad at round } i) \\
&\leq \sum_{\substack{\text{tree } T \subseteq G_{i-1}^5 \\ |T| = s/\Delta^4}} \sum_{\substack{B_1, \ldots, B_{c_1 \ln \Delta} \subseteq T \\ \bigcup B_i = T, B_i \cap B_j = \emptyset}} \prod_i p^{|B_i|} \\
&= \sum_{\substack{\text{tree } T \subseteq G_{i-1}^5 \\ |T| = s/\Delta^4}} \sum_{\substack{B_1, \ldots, B_{c_1 \ln \Delta} \subseteq T \\ \bigcup B_i = T, B_i \cap B_j = \emptyset}} p^{s/\Delta^4} \\
&\leq n \left((e\Delta^5)(c_1 \ln \Delta) p \right)^{s/\Delta^4} \\
&\leq n(1/2)^{s/\Delta^4}
\end{aligned}
$$

which is at most $1/\operatorname{poly}(n)$, if $s = \Omega(\Delta^4 \log n)$. Therefore, with high probability, all bad components have size at most $O(\Delta^4 \log n)$.

3.5 Conclusion

The time bounds of Theorem 3.1.1 show an interesting discontinuity. When Δ is large we can cap the error at $1/\operatorname{poly}(n)$ by using standard concentration inequalities and a union

bound. When Δ is small we can use our algorithm for LLL to reduce the failure probability again to $1/\operatorname{poly}(n)$.

We showed that $\chi(G) \leq (4+o(1))\Delta/\ln\Delta$ for triangle-free graphs G. It would be interesting to see if it is possible to reduce the palette size to $(1+o(1))\Delta/\ln\Delta$, matching Kim's [104] bound for girth-5 graphs.

3.6 Jamall's Analysis

There is a small flaw in Jamall's proof of Lemma 12 in [86], the corresponding Lemma 17 in [87], and the corresponding lemmas in [88]. He defined the following quantities:

$d_t(u,c)$: the c-degree of u at the beginning of round t, which corresponds to $|N_{t-1,c}(u)|$ in our case.

$S_t(u)$: the palette of u at the beginning of round t, which corresponds to $P_{t-1}(u)$ in our case. Also, he defined $s_t(u) = |S_t(u)|$.

$\tilde{d}_t(u,c)$: the c-degrees of u just before the cleanup phase (filtering out colors whose c-degrees are too large) of round t, which corresponds to $|N_{t,c}(u)|$ in our case.

$\bar{d}_t(u) := \sum_{c\in\tilde{S}_t(u)} \tilde{d}_t(u,c)$, which corresponds to $\widehat{n}_t(u)\cdot|\widehat{P}_t(u)|$ in our case.

$\tilde{S}_t(u)$: the palette of u just before the cleanup phase of round t, which corresponds to $\widehat{P}_t(u)$ in our case.

In [86, p. 13]:

> For concentration of $\bar{d}_t(u)$, suppose $s_t(u) = m$. Let $c_1,\ldots,c_m$ be the colors in $S_t(u)$. Then $\bar{d}_t(u)$ may be considered a random variable determined by the random trials $T_1,\ldots,T_m$, where T_i is the set of vertices in G_t that are assigned color c_i in round t. Observe that T_i affects $\bar{d}_t(u)$ by at most $d_t(u,c)$.

He claimed that each of the random trials T_i only affects $\bar{d}_t(u)$ by $d_t(u,c)$, which is the range of the term $\tilde{d}_t(u,c)$ (i.e. $\tilde{d}_t(u,c)\in[0,d_t(u,c)]$) in the sum $\bar{d}_t(u) = \sum_{c\in\tilde{S}_t(u)}\tilde{d}_t(u,c)$. However,

this is not necessarily true, since it is possible that a single exposure of T_i can cause all c-neighbors to become colored. This may affect more than one term in the sum and thus more than the amount of $d_t(u, c)$.

For example, at the initial configuration, where each vertex has the same palette, the c-degree of u, $d_t(u, c)$, are equal for all colors c. Suppose that after exposing $T_1, \ldots, T_{m-1}$, we have $T_1 = \cdots = T_{m-1} = \emptyset$. When we expose T_m, if T_m is also an emptyset, then $\bar{d_t}(u) = \sum_{i=1}^{m} d_t(u, c_i)$. On the other hand, if T_m is exactly the neighbor set of u, then $\bar{d_t}(u) = 0$, because every neighbor becomes colored. The difference can be as large as $\sum_{i=1}^{m} d_t(u, c_i) = m d_t(u, c_m)$ rather than claimed $d_t(u, c_m)$. This bound is too large to apply Azuma's inequality, because $\sum \alpha_i^2$ in their proof can become as large as $O(s_t^2(u) d_t^2(u))$. Perhaps it is possible to fix it by bounding the unlikely events or by considering the average difference rather then just considering the absolute difference. We presented a different analysis in this chapter, whose concentration bound also satisfies the demands of an efficient distributed implementation.

Chapter 4

Edge Coloring

4.1 Introduction

In this chapter, we focus on the $(1+\epsilon)\Delta$-edge-coloring problem in the distributed setting. We devise a drastically improved algorithm for $(1+\epsilon)\Delta$-edge-coloring. Using the Rödl nibble method Dubhashi, Grable, and Panconesi [39] devised a $(1+\epsilon)\Delta$-edge-coloring algorithm for graphs with $\Delta = (\log n)^{1+\Omega(1)}$ which requires $O(\log n)$ time. In this chapter we devise a $(1+\epsilon)\Delta$-edge-coloring algorithm for graphs with $\Delta \geq \Delta_\epsilon$ (Δ_ϵ is a constant that depends on ϵ.) with running time $O(\log^* \Delta \cdot \max\{1, \frac{\log n}{\Delta^{1-o(1)}}\})$. In particular, for $\Delta = (\log n)^{1+\Omega(1)}$ the running time of our algorithm is only $O(\log^* n)$, as opposed to the previous state-of-the-art of $O(\log n)$ [39]. For smaller values of Δ, we will use our distributed algorithm for LLL from Chapter 2.

As a byproduct of the algorithm, we obtain the first *sublogarithmic* time algorithm for the $(2\Delta-1)$-edge-coloring problem. Specifically, our algorithm requires $\exp(O(\sqrt{\log \log n}))$ time, i.e., less than $\log^\epsilon n$ time for any $\epsilon > 0$. (In particular, it is far below the $\Omega(\sqrt{\log n})$ barrier of [110].) Therefore, our result establishes a clear separation between the complexities of the $(2\Delta-1)$-edge-coloring and MM problems.

4.1.1 Related Work

Our algorithm is randomized. The study of distributed randomized edge-coloring was initiated by Panconesi and Srinivasan [142]. The result of [142] was later improved in the aforementioned paper of [39].

Significant research attention was also devoted to deterministic edge-coloring algorithms, but those typically use much more than $(1+\epsilon)\Delta$ colors. Specifically, Panconesi and Rizzi gave a deterministic $(2\Delta-1)$-edge-coloring algorithm that runs in $O(\Delta+\log^* n)$ rounds [140]. Czygrinow et al. [30] devised a deterministic $O(\Delta \cdot \log n)$-edge-coloring algorithm with running time $O(\log^4 n)$. More recently Barenboim and Elkin [9] devised a deterministic $O(\Delta^{1+\epsilon})$-edge-coloring algorithm with running time $O(\log \Delta + \log^* n)$, and an $O(\Delta)$-edge-coloring algorithm with time $O(\Delta^{\epsilon} + \log^* n)$, for an arbitrary small $\epsilon > 0$.

4.1.2 Technical Overview

We begin by discussing the $(1+\epsilon)\Delta$-edge coloring problem. Our algorithm consists of multiple rounds that color the edges of the graph gradually. Let $P(u)$ denote the palette of u, which consists of colors not assigned to the edges incident to u. Therefore, an edge uv can choose a color from $P(uv) \stackrel{\text{def}}{=} P(u) \cap P(v)$. Our goal is to show that $P(uv)$ will always be non-empty as the algorithm proceeds and we hope to color the graph as fast as possible. If $P(u)$ and $P(v)$ behave like independent random subsets out of the $(1+\epsilon)\Delta$ colors, then the expected size of $P(uv)$ is at least $(\epsilon/(1+\epsilon))^2 \cdot (1+\epsilon)\Delta$, since the size of $P(u)$ and $P(v)$ is $\epsilon/(1+\epsilon)$ fraction of the original palette. This means if the size of $P(uv)$ concentrates around its expectation, then it will be non-empty.

We use the following process to color the graph while keeping the palettes behaving randomly. In each round, every edge selects a set of colors in its palette. If an edge selected a color that is not selected by adjacent edges, then it will become colored with one such color. The colored edges will be removed from the graph.

In contrast with the framework of [39, 73], where each edge selects at most one color in each round, selecting multiple colors allows us to break symmetry faster. The idea of selecting multiple colors independently has been used in [88,172] to reduce the dependency introduced

in the analysis for triangle-free graphs and locally-sparse graphs. Our analysis is based on the semi-random method or the so-called Rödl Nibble method, where we show by induction that after each round a certain property H_i holds w.h.p., assuming H_{i-1} holds. In particular, H_i is the property that the palette size of each edge is lower bounded by p_i, and the c-degree of a vertex, that is, the number of uncolored adjacent edges having the color c in its palette, is upper bounded by t_i. Intuitively, the symmetry is easier to break when the size of the palette is larger and when the c-degree is smaller. Therefore, we hope that the probability an edge becomes colored increases with p_i/t_i. By selecting multiple colors for each edge in each round, we will capture this intuition and be able to color the graph faster than by selecting just one single color.

The main technical challenge is to prove the concentration bounds. To this end, we use exisiting techniques and develop new techniques to minimize the dependencies introduced. First, we use the *wasteful coloring procedure* [125]: Instead of removing colors from the palette that are colored by the neighbors, we remove the colors that are selected by the neighbors in each round. In this way, we can zoom in the analysis into the 2-neighborhood of a vertex instead of 3-neighborhood. Also, we use the expose-by-ID-ordering technique introduced in [145]. In the edge coloring problem, assume that each edge has a unique ID. In each round, we let an edge become colored if it selected a color that is not selected by its neighbor with smaller ID. Therefore, the choices of the neighbors with larger ID will not affect the outcome of the edge. That makes bounding the difference or the variance of the martingales much simpler when we expose the choices of the edges according to the order of their ID. Finally, we derive a modification of Chernoff Bound (Lemma A.5) that is capable to handle the sum of non-independent random variables conditioned on some likely events. In particular, although the expectation of the i-th random variable may be heavily affected by the configuration of first $i-1$ random variables, our inequality applies if we can bound the expectation when conditioning on some very likely events that depend on the first $i-1$ random variables. When combined with the expose-by-ID-ordering technique, it becomes a useful tool for the analysis of concentration. (See the proofs of Lemma 4.2.6.)

4.2 Distributed Edge Coloring

Given a graph $G = (V, E)$, we assume each edge e has a unique identifier, $\mathrm{ID}(e)$. For each edge, we maintain a palette of available colors. Our algorithm proceeds by rounds. In each round, we color some portion of the graph and then delete the colored edges. Let G_i be the graph after round i and $P_i(e)$ be the palette of e after round i. Initially, $P_0(e)$ consist of all the colors $\{1, 2, \ldots, (1+\epsilon)\Delta\}$. We define the sets $N_i(\cdot) : V \cup E \to 2^E$, $N_{i,c}(\cdot) : V \cup E \to 2^E$, and $N^*_{i,c}(e) : E \to 2^E$ as follows. $N_i(\cdot)$ is the set of neighboring edges of a vertex or an edge in G_i. $N_{i,c}(\cdot)$ is the set of neighboring edges of a vertex or an edge in G_i having c in its palette. $N^*_{i,c}(e)$ is the set of neighboring edges having smaller ID than e and having c in its palette in G_i.

For clarity we use the following shorthands: $\deg_i(\cdot) = |N_i(\cdot)|$, $\deg_{i,c}(\cdot) = |N_{i,c}(\cdot)|$, and $\deg^*_{i,c}(e) = |N^*_{i,c}(e)|$, where $\deg_{i,c}(\cdot)$ is often referred as the *c-degree*. Also, if $F(\cdot)$ is a set function and S is a set, we define $F(S) = \bigcup_{s \in S} F(s)$.

Theorem 4.2.1. *Let $\epsilon, \gamma > 0$ be constants. There exists a constant $\Delta_{\epsilon,\gamma} \geq 0$ and a distributed algorithm such that for all graphs with $\Delta \geq \Delta_{\epsilon,\gamma}$, the algorithm colors all the edges with $(1+\epsilon)\Delta$ colors and runs in $O(\log^* \Delta \cdot \max(1, \log n/\Delta^{1-\gamma}))$ rounds.*

Corollary 4.2.2. *For any Δ, the $(2\Delta - 1)$-edge-coloring problem can be solved in $\exp(O(\sqrt{\log \log n}))$ rounds.*

Proof. Let $\epsilon = 1$ and $\gamma = 1/2$. By Theorem 4.2.1, there exists a constant $\Delta_{1,1/2}$ such that for $\Delta \geq \max((\log n)^2, \Delta_{1,1/2})$, the problem can be solved in $O(\log^* \Delta)$ rounds. Otherwise $\Delta = O(\log^2 n)$ and we can apply the $(\Delta+1)$-vertex coloring algorithm in [11] to the line graph of G, which takes $O(\log \Delta + \exp(O(\sqrt{\log \log n}))) = \exp(O(\sqrt{\log \log n}))$ rounds. □

We describe the algorithm of Theorem 4.2.1 in Algorithm 9 for $\Delta > (\log n)^{1/(1-\gamma)}$. In the end of the section, we show how to generalize it to smaller Δ by using a distributed algorithm for contructive Lovász Local Lemma [23]. The algorithm proceeds in rounds. We will define $\{\pi_i\}$ and $\{\beta_i\}$ later. For now, let us think π_i is inversely proportional to the c-degrees and β_i is a constant.

In each round i, each edge e selects two set of colors $S_i(e)$ and $K_i(e)$ by using Algorithm 10. $S_i(e)$ is selected by including each color in $P_{i-1}(e)$ with probability π_i independently. The

Edge-Coloring-Algorithm $(G, \{\pi_i\}, \{\beta_i\})$

$G_0 \leftarrow G$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  for each $e \in G_{i-1}$ **do**
    $(S_i(e), K_i(e)) \leftarrow \text{Select}(e, \pi_i, \beta_i)$
    Set $P_i(e) \leftarrow K_i(e) \setminus S_i(N^*_{i-1}(e))$
    if $S_i(e) \cap P_i(e) \neq \emptyset$ **then** color e with any color in $S_i(e) \cap P_i(e)$ **end if**
  end for
  $G_i \leftarrow G_{i-1} \setminus \{\text{colored edges}\}$
until

Algorithm 9

Select(e, π_i, β_i)

Include each $c \in P_{i-1}(e)$ in $S_i(e)$ independently with probability π_i.
For each c, calculate $r_c = \beta_i^2/(1-\pi_i)^{\deg^*_{i-1,c}(e)}$.
Include $c \in P_{i-1}(e)$ in $K_i(e)$ independently with probability r_c.
return $(S_i(e), K_i(e))$.

Algorithm 10

colors selected by the neighbors with smaller ID than e, $S_i(N^*_{i-1}(e))$, will be removed from e's palette. To make the analysis simpler, we would like to ensure that each color is removed from the palette with an identical probability. Thus, $K_i(e)$ is used for this purpose. A color c remains in $P_i(e)$ only if it is in $K_i(e)$ and no neighboring edge with smaller ID selected c. The probability that this happens is exactly $(1-\pi_i)^{\deg^*_{i-1,c}(e)} \cdot r_c = \beta_i^2$. Note that r_c is always at most 1 if $\deg^*_{i-1,c}(u) \leq t'_{i-1}$ (defined below), which we later show holds by induction. An edge will become colored if it has selected a color remaining in $P_i(e)$. Obviously, no two adjacent edges will be colored the same in the process.

We will assume Δ is sufficiently large whenever we need certain inequalities to hold. The asymptotic notations are functions of Δ. Let $p_0 = (1+\epsilon)\Delta$ and $t_0 = \Delta$ be the initial lower bound on the palette size and initial upper bound on the c-degree of a vertex. Let

$$\begin{aligned}
\pi_i &= 1/(Kt'_{i-1}) & \delta &= 1/\log\Delta \\
\alpha_i &= (1-\pi_i)^{p'_i} & \beta_i &= (1-\pi_i)^{t'_{i-1}-1} \\
p_i &= \beta_i^2 p_{i-1} & t_i &= \max(\alpha_i\beta_i t_{i-1}, T) \\
p'_i &= (1-\delta)^i p_i & t'_i &= (1+\delta)^{2i} t_i \\
K &= 4 + 4/\epsilon & T &= \Delta^{1-0.9\gamma}/2
\end{aligned}$$

p_i and t_i are the ideal (that is, expected) lower and upper bounds of the palette size and the vertex c-degrees after round i. p'_i and t'_i are the relaxed version of p_i and t_i with error $(1-\delta)^i$ and $(1+\delta)^{2i}$, where δ is chosen to be small enough such that $(1-\delta)^i = 1-o(1)$ and $(1+\delta)^{2i} = 1+o(1)$ for all i we consider, i.e. for $i = O(\log^*\Delta)$.

π_i is the sampling probability in our algorithm. We will show that α_i is an upper bound on the probability an edge remains uncolored in round i and β_i^2 is the probability a color

remains in the palette of an edge depending on ϵ. Since

$$\begin{aligned}
\beta_i &= \left(1 - \frac{1}{(Kt'_{i-1} - 1) + 1}\right)^{(Kt'_{i-1}-1)\cdot\frac{t'_{i-1}-1}{Kt'_{i-1}-1}} \\
&\geq \left(1 - \frac{1}{(Kt'_{i-1} - 1) + 1}\right)^{(Kt'_{i-1}-1)\cdot\frac{1}{K}} \\
&\geq e^{-1/K}. \qquad \text{Since } \left(1 - \tfrac{1}{x+1}\right)^x \geq e^{-1}.
\end{aligned}$$

Therefore, β_i is bounded below by $e^{-1/K}$, which is a constant. While p_i shrinks by β_i^2, we will show t_i shrinks by roughly $\alpha_i\beta_i$. Note that $p_0/t_0 \geq (1+\epsilon)$ initially. The constant K is chosen so that $e^{-2/K}(1+\epsilon) - 1 = \Omega(\epsilon)$ and so α_i is smaller than β_i initially, since we would like to have t_i shrink faster than p_i. Then, α_i becomes smaller as the ratio between t_i and p_i becomes smaller. Finally, we cap t_i by T, since our analysis in the first phase does not have strong enough concentration when t_i decreases below this threshold. Thus, we will switch to the second phase, where we trade the amount t_i decreases (which is supposed to be decreased to its expectation as in the first phase) for a smaller error probability.

We will show that the first phase ends in $O(\log^* \Delta)$ rounds and the second phase ends in a constant number of rounds. We will discuss the number of rounds in the second phase later in this section.

Lemma 4.2.3. *$t_r = T$ after at most $r = O(\log^* \Delta)$ rounds.*

Proof. We divide the process into two stages. The first is when $t_{i-1}/p_{i-1} \geq 1/(1.1e^{3/K}K)$.

In this stage,

$$
\begin{aligned}
&\frac{t_i}{p_i} = \frac{\alpha_i}{\beta_i}\frac{t_{i-1}}{p_{i-1}} \\
&= (1-\pi_i)^{p'_i - t'_{i-1}+1} \cdot \frac{t_{i-1}}{p_{i-1}} && \text{defn. } \alpha_i, \beta_i \\
&\le \exp\left(-\pi_i \cdot (p'_i - t'_{i-1} + 1)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && 1 - x \le e^{-x} \\
&\le \exp\left(-(1-o(1)) \cdot \frac{1}{K}\left(\frac{p_i}{t_{i-1}} - 1\right)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && \text{defn. } \pi_i,\ \frac{p'_i}{t'_{i-1}} = (1-o(1))\frac{p_i}{t_{i-1}} \\
&\le \exp\left(-(1-o(1)) \cdot \frac{1}{K}\left(\frac{\beta_i^2 p_{i-1}}{t_{i-1}} - 1\right)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && \text{defn. } p_i \\
&\le \exp\left(-(1-o(1)) \cdot \frac{1}{K}\left(e^{-2/K}(1+\epsilon) - 1\right)\right) \cdot \frac{t_{i-1}}{p_{i-1}} && p_{i-1}/t_{i-1} \ge (1+\epsilon) \\
&\le \exp\left(-(1-o(1)) \cdot \frac{((1-2/K)(1+\epsilon) - 1)}{K}\right) \cdot \frac{t_{i-1}}{p_{i-1}} && e^{-x} \ge 1 - x \\
&= \exp\left(-(1-o(1)) \cdot \frac{\epsilon^2}{8(1+\epsilon)}\right) \cdot \frac{t_{i-1}}{p_{i-1}} && K = 4(1+\epsilon)/\epsilon
\end{aligned}
$$

Therefore, after at most $(1+o(1))\frac{8(1+\epsilon)}{\epsilon^2}\ln\left(1.1Ke^{3/K}\right)$ rounds, this stage will end. Let j be the first round when the second stage starts. For $i > j$, we have

$$
\begin{aligned}
&\alpha_i = (1-\pi_i)^{p'_i} \\
&\le \exp\left(-(1-o(1))\frac{1}{K} \cdot \frac{p_i}{t_{i-1}}\right) && 1 - x \le e^{-x} \\
&\le \exp\left(-(1-o(1))\frac{1}{K} \cdot \frac{\beta_i^2 p_{i-1}}{t_{i-1}}\right) && \text{defn. } p_i \\
&\le \exp\left(-(1-o(1))\frac{1}{K} \cdot \frac{\beta_{i-1}}{\alpha_{i-1}} \cdot \frac{\beta_i^2 p_{i-2}}{t_{i-2}}\right) && \frac{p_{i-1}}{t_{i-1}} = \frac{\beta_{i-1}}{\alpha_{i-1}}\frac{p_{i-2}}{t_{i-2}} \\
&\le \exp\left(-(1-o(1))\frac{1}{K} \cdot \frac{e^{-3/K}}{\alpha_{i-1}} \cdot \frac{p_{i-2}}{t_{i-2}}\right) && \beta_i \ge e^{-1/K} \\
&\le \exp\left(-1/\alpha_{i-1}\right) && \frac{t_{i-2}}{p_{i-2}} < \frac{1}{1.1Ke^{3/K}}
\end{aligned}
$$

Therefore, $\frac{1}{\alpha_{j+\log^* \Delta+1}} \geq \underbrace{e^{e^{\cdot^{\cdot^{e}}}}}_{\log^* \Delta} \geq \Delta$, and so $t_{j+\log^* \Delta+1} \leq \max(\alpha_{j+\log^* \Delta+1} \cdot \Delta, T) = T$. □

Then, we show the bound on the palette size remains large throughout the algorithm.

Lemma 4.2.4. *$p'_i = \Delta^{1-o(1)}$ for $i = O(\log^* \Delta)$.*

Proof. $p'_i = (1-\delta)^i p_i \geq (1-\delta)^i \prod_{j=1}^{i} \beta_j^2 \Delta \geq (1-\delta)^i e^{-2i/K} \Delta = (1-o(1))\Delta^{-\frac{2i}{K \log \Delta}} \cdot \Delta = \Delta^{1-o(1)}$. □

Let $H_i(e)$ denote the event that $|P_i(e)| \geq p'_i$ and $H_{i,c}(u)$ denote the event $\deg_{i,c}(u) \leq t'_i$. Let H_i be the event such that for all $u, e \in G$ and all $c \in P_i(u)$, $H_{i,c}(u)$ and $H_i(e)$ hold. Supposing that H_{i-1} is true, we will estimate the probability that $H_i(e)$ and $H_{i,c}(u)$ are true.

Lemma 4.2.5. *Suppose that H_{i-1} is true, then* $\Pr(|P_i(e)| < (1-\delta)\beta_i^2 |P_{i-1}(e)|) < e^{-\Omega(\delta^2 p'_i)}$.

Proof. Consider a color $c \in P_{i-1}(e)$. The probability c remains in $P_i(e)$ is exactly $\Pr(c \notin S_i(N^*_{i-1}(e))) \cdot \Pr(c \in K_i(e)) = \beta_i^2$. Since the event that c remains in the palette is independent among other colors, by a Chernoff bound, $\Pr(|P_i(e)| < (1-\delta)\beta_i^2 |P_{i-1}(e)|) < e^{-\Omega(\delta^2 p'_i)}$. □

Lemma 4.2.6. *Suppose that H_{i-1} is true, then* $\Pr(\deg_{i,c}(u) > t'_i) < 2e^{-\Omega(\delta^2 T)} + \Delta e^{-\Omega(\delta^2 p'_i)}$.

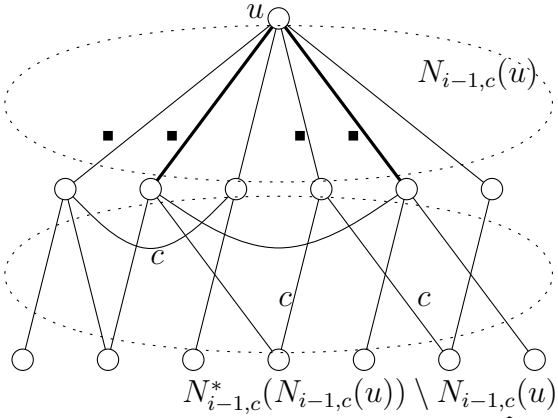
Proof. Define the auxiliary set

$$\widehat{N}_{i,c}(u) \stackrel{\text{def}}{=} \{e \in N_{i-1,c}(u) \mid (c \in K_i(e)) \text{ and } (c \notin S(N^*_{i-1}(e) \setminus N_{i-1,c}(u)))\}$$

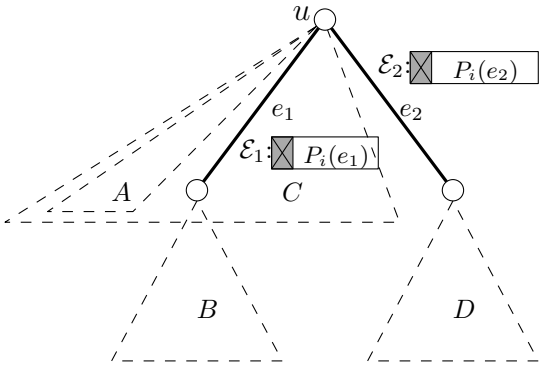
and $\widehat{\deg}_{i,c}(u) = |\widehat{N}_{i,c}(u)|$ (see Figure 2a). $\widehat{N}_{i,c}(u)$ is the set of edges $uv \in N_{i-1,c}(u)$ that keep the color c in $K_i(uv)$ and no edges adjacent to v (except possibly uv) choose c. We will first show that $\Pr(\widehat{\deg}_{i,c}(u) \leq (1+\delta)\beta_i \deg_{i-1,c}(u)) \leq e^{-\Omega(\delta^2 T)}$. Consider $e = uv \in N_{i-1,c}(u)$. The probability that $c \in K_i(e)$ and $c \notin S(N^*_{i-1}(e) \setminus N_{i-1,c}(u))$ both happen is

$$\begin{aligned}
&\frac{(1-\pi_i)^{2t'_{i-1}-2}}{(1-\pi_i)^{\deg^*_{i-1,c}(v)+\deg^*_{i-1,c}(u)-2}} \cdot (1-\pi_i)^{\deg^*_{i-1,c}(v)-1} \\
&\leq \frac{(1-\pi_i)^{t'_{i-1}-1}}{(1-\pi_i)^{\deg^*_{i-1,c}(v)-1}} \cdot (1-\pi_i)^{\deg^*_{i-1,c}(v)-1} = \beta_i.
\end{aligned}$$

Figure 2



(a) The bold lines denote the edges in $\widehat{N}_{i,c}(u)$. In this example, we assume all the edges in the bottom have smaller ID than the edges on the top. The solid square besides an edge e in the top part denote that $c \in K_i(e)$. The character 'c' besides an edge e in the bottom part denote that $c \in S_i(e)$. The set $\widehat{N}_{i,c}(u)$ is determined by the squares and the c's.



(b) An illustration showing the probability that e_2 selects a color $c' \in P_i(e_2)$ is unaffected when conditioning on $\mathcal{E}_1$, $\mathcal{E}_2$, and whether e_1 is colored or not. Note that $e_1, e_2 \in \widehat{N}_{i-1,c}(u)$ and $\mathrm{ID}(e_1) < \mathrm{ID}(e_2)$. $\mathcal{E}_1$ is a function of $K_i(e_1)$ and the colors chosen by the edges in A and B. $\mathcal{E}_2$ is a function of $K_i(e_2)$ and the colors chosen by the edges in C and D. Thus, conditioning on them does not affect the probability e_2 select c'. Furthermore, whether e_1 is colored does not depend on whether e_2 selects the colors in $P_i(e_2)$, but only possibly depends on whether the colors in the grey area $(P_{i-1}(e_2) \setminus P_i(e_2))$ are selected.

Let $e_1, \ldots, e_k$ be the edges in $N_{i-1,c}(u)$ and let $e'_1, \ldots, e'_{k'}$ be the edges in $N^*_{i-1,c}(N_{i-1,c}(u)) \setminus N_{i-1,c}(u)$. Clearly, $\widehat{\deg}_{i,c}(u)$ is determined solely by $K_i(e_1), \ldots, K_i(e_k)$ and $S_i(e'_1), \ldots, S_i(e'_{k'})$.

Define the following sequence:

$$\boldsymbol{Y}_j = \begin{cases} \emptyset & j = 0 \\ (K_i(e_1), \ldots, K_i(e_j)) & 1 \le j \le k \\ \left(\boldsymbol{Y}_k, S_i(e'_1), \ldots, S_i(e'_{j-k})\right) & k < j \le k + k' \end{cases}$$

Let V_j be

$$\mathrm{Var}\left(\mathrm{E}[\widehat{\deg}_{i,c}(u) \mid \boldsymbol{Y}_{j-1}] - \mathrm{E}[\widehat{\deg}_{i,c}(u) \mid \boldsymbol{Y}_j] \;\middle|\; \boldsymbol{Y}_{j-1}\right).$$

We will upper bound V_j and apply the concentration inequalities of Lemma A.8. For $1 \le j \le k$, the exposure of $K_i(e_j)$ affects $\widehat{\deg}_{i,c}(u)$ by at most 1, so $V_j \le 1$ and $\sum_{1\le j\le k} V_j \le t'_{i-1}$. For $k < j \le k+k'$, the exposure of $S_i(e_j)$ affects $\widehat{\deg}_{i,c}(u)$ by at most 2, since edge e'_j is adjacent to at most 2 edges in $N_{i-1,c}(u)$. Since the probability e_j selects c is π_i, $V_j \le 4\pi_i$. (We make a query about whether c is contained in $S_i(e_j)$. For an yes/no query, the variance is bounded by $p_{\text{yes}} \cdot C^2$, if the function is C-Lipschitz and p_{yes} is the probability that the answer to the query is yes [39, 40].) Therefore, $\sum_{k<j\le k+k'} V_j \le 4k'\pi_i \le 4t'^2_{i-1}\pi_i = 4t'_{i-1}/K \le 4t'_{i-1}$. The total variance, $\sum_{1\le j\le k+k'} V_j$, is at most $5t'_{i-1}$.

We apply Lemma A.8 with $M = 2$, $t = \delta\beta_i t'_{i-1}$, and $\sigma_j^2 = V_j$ to get

$$
\begin{aligned}
&\Pr(\widehat{\deg}_{i,c}(u) > (1+\delta)\beta_i t'_{i-1}) \\
&\le \Pr(\widehat{\deg}_{i,c}(u) > \beta_i \deg_{i-1,c}(u) + t) && \deg_{i-1,c}(u) \le t'_{i-1} \\
&\le \exp\left(-\frac{t^2}{2(\sum_{j=1}^{k+k'} \sigma_j^2 + 2t/3)}\right) \\
&= \exp\left(-\frac{t^2}{2(5t'_{i-1} + 2t/3)}\right) \\
&\le \exp\left(-\frac{\delta^2\beta_i^2 t'^2_{i-1}}{2(5t'_{i-1} + 2(\delta\beta_i t'_{i-1})/3)}\right) \\
&= \exp\left(-\Omega(\delta^2 t'_{i-1})\right)
\end{aligned}
$$

Next, we show $\Pr(\deg_{i,c}(u) > (1+\delta)\alpha_i \widehat{\deg}_{i,c}(u)) \le \Delta e^{-\Omega(\delta^2 p'_i)} + e^{-\Omega(\delta^2 T)}$. Let $e_1, \ldots, e_k \in \widehat{N}_{i,c}(u)$ listed by their ID in increasing order. Let $\mathcal{E}_j$ denote the likely event that $|P_i(e_j)| \ge p'_i$. Notice that $\Pr(c \in P_i(e_j) \mid e_j \in \widehat{N}_{i,c}(u)) \ge \Pr(c \in P_i(e_j)) \ge \beta_i$ and $\Pr(c' \in P_i(e_j) \mid e_j \in \widehat{N}_{i,c}(u)) = \Pr(c' \in P_i(e_j)) \ge \beta_i$ for all other $c' \ne c$ and $c' \in P_{i-1}(e_j)$. Therefore, $\mathrm{E}[|P_i(e_j)| \mid e_j \in \widehat{N}_{i,c}(u)] \ge \beta_i |P_{i-1}(e_j)| \ge \beta_i p'_{i-1}$.

By Lemma 4.2.5, $\Pr(\overline{\mathcal{E}_j}) \le e^{-\Omega(\delta^2 p'_i)}$. Let X_j be the event that e_j is not colored after this round and let $\boldsymbol{X}_j$ be the shorthand for $(X_1, \ldots, X_j)$. We will show that

$$\max_{\boldsymbol{X}_{j-1}} \Pr(X_j \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \le \alpha_i$$

and so we can apply Lemma $A.5$, a Chernoff-type tail bound when conditioning on a sequence of very likely events. First, we argue that for any $\boldsymbol{X}_{j-1}$ and $c' \in P_i(e_j)$, $\Pr(c' \in S_i(e_j) \mid$

$\boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) = \pi_i$ (see Figure 2b). Since $c' \in P_i(e_j)$, c' is not chosen by any of the edges $e_1, e_2, \ldots, e_{j-1}$. Therefore, whether these edges become colored does not depend on whether they choose c' or not. Furthermore, conditioning on $\mathcal{E}_1, \ldots, \mathcal{E}_j$ has no effect on the probability e_j selects c', because the palette sizes of $e_1, \ldots e_j$ do not depend on the colors chosen by e_j, but only the choices of the edges with smaller ID. Therefore, we have:

$$\begin{aligned}
&\Pr(X_j \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \\
&= \prod_{c' \in P_i(e_j)} \Pr(c' \notin S_i(e_j) \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \\
&= (1-\pi_i)^{|P_i(e_j)|} \leq (1-\pi_i)^{p'_i} && \mathcal{E}_j \text{ is true} \\
&= \alpha_i
\end{aligned}$$

Notice that by Lemma 4.2.5, $\sum_j \Pr(\overline{\mathcal{E}}_j) \leq \Delta e^{-\Omega(\delta^2 p'_i)}$. By Lemma A.5 and Corollary A.2, we have:

$$\begin{aligned}
&\Pr(\deg_{i,c}(u) > \alpha_i \cdot \widehat{\deg}_{i,c}(u) + \delta \max(\alpha_i \cdot \widehat{\deg}_{i,c}(u), T)) \\
&\leq e^{-\Omega(\delta^2 T)} + \Delta e^{-\Omega(\delta^2 p'_i)}
\end{aligned}$$

By the union bound, the probability that both $\deg_{i,c}(u) \leq \alpha_i \cdot \widehat{\deg}_{i,c}(u) + \delta \max(\alpha_i \cdot \widehat{\deg}_{i,c}(u), T)$ and $\widehat{\deg}_{i,c}(u) \leq (1+\delta)\beta_i t'_{i-1}$ hold is at least $1 - 2e^{-\Omega(\delta^2 T)} - \Delta e^{-\Omega(\delta^2 p'_i)}$. When both of them are true:

$$\begin{aligned}
&\deg_{i,c}(u) \\
&\leq (1+\delta)\alpha_i\beta_i t'_{i-1} + \delta \max((1+\delta)\alpha_i\beta_i t'_{i-1}, T) \\
&\leq (1+\delta)\alpha_i\beta_i t'_{i-1} + \delta \max((1+\delta)\alpha_i\beta_i t'_{i-1}, t_i) && T \leq t_i \\
&\leq (1+\delta)\alpha_i\beta_i t'_{i-1} + \delta(1+\delta)^{2i-1} t_i \leq t'_i && \text{defn. } t_i \text{ and } t'_i
\end{aligned}$$

□

Second Phase Suppose that H_r holds at the end of iteration r, where r is the first round where $t_r = T$ and so $\deg_{r,c}(u) \leq t'_r \leq 2T$ for all u and c. Now we will show the algorithm terminates in a constant number of rounds. For $i > r$, let $t'_i = t'_{i-1} \cdot \frac{T}{p'_i}$.

Recall that $H_i(e)$ denotes the event that $|P_i(e)| \geq p'_i$ and $H_{i,c}(u)$ denotes the event that $\deg_{i,c}(u) \leq t'_i$ (Notice that t'_i has a different definition when $i > r$ than that when $0 \leq i \leq r$). Also recall H_i denotes the event that $H_i(e)$ and $H_{i,c}(u)$ are true for all $u, e \in G_i$ and all $c \in P_i(u)$. If Δ is large enough, then we can assume that $p'_i \geq \Delta^{1-0.8\gamma}$ by Lemma 4.2.4. Then from the definition of t'_i, it shrinks to less than one in $\lceil \frac{1}{0.1\gamma} \rceil$ rounds, since $T/p'_i \leq \Delta^{-0.1\gamma}$ and $t'_{r+1/(0.1\gamma)} < (\Delta^{-0.1\gamma})^{\lceil 1/(0.1\gamma) \rceil} \cdot t'_r < 1$.

Suppose that H_{i-1} is true, we will estimate the probability that $H_i(e)$ and $H_{i,c}(u)$ are true. Consider a color $c \in P_{i-1}(e)$. It is retained in the palette with probability exactly β_i^2, so $\mathrm{E}[|P_i(e)|] \geq \beta_i^2 |P_{i-1}(e)| \geq \beta_i^2 p'_{i-1}$. Since each color is retained in the palette independently, by a Chernoff Bound, $\Pr(|P_i(e)| < (1-\delta)\beta_i^2 \cdot p'_{i-1}) < e^{-\Omega(\delta^2 p'_i)}$.

Lemma 4.2.7. *Suppose that H_{i-1} is true where $i > r$, then* $\Pr(\deg_{i,c}(u) > t'_i) < e^{-\Omega(T)} + \Delta e^{-\Omega(\delta^2 p'_i)}$.

Proof. We will now bound the probability that $\deg_{i,c}(u) > t'_i$. Let $e_1, \ldots, e_k \in N_{i-1,c}(u)$, listed by their ID in increasing order. Let $\mathcal{E}_j$ denote the likely event that $|P_i(e_j)| \geq p'_i$. Notice that $\Pr(\overline{\mathcal{E}}_j) \leq e^{-\Omega(\delta^2 p'_i)}$ by Lemma 4.2.5. For each $e_j \in N_{i,c}(u)$, let X_j denote the event that e_j is not colored. As we have shown previously $\Pr(X_j \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \leq \alpha_i$, therefore,

$$\begin{aligned}
&\Pr(\deg_{i,c}(u) > t'_i) \\
&= \Pr\left(\deg_{i,c}(u) > \left(\frac{t'_i}{\alpha_i t'_{i-1}}\right) \cdot \alpha_i t'_{i-1}\right).
\end{aligned}$$

Applying Lemma A.5 and Corollary A.2 with $1+\delta = t'_i/(\alpha_i t'_{i-1})$, and noticing that $\alpha_i \deg_{i-1,c}(u) \le \alpha_i t'_{i-1}$, the probability above is bounded by

$$
\begin{aligned}
&\exp\left(-\alpha_i t'_{i-1}\left(\frac{t'_i}{\alpha_i t'_{i-1}}\ln\frac{t'_i}{\alpha_i t'_{i-1}} - \left(\frac{t'_i}{\alpha_i t'_{i-1}} - 1\right)\right)\right) + \Delta e^{-\Omega(\delta^2 p'_i)} \\
&\le \exp\left(-t'_i\left(\ln\frac{t'_i}{\alpha_i t'_{i-1}} - 1\right)\right) + \Delta e^{-\Omega(\delta^2 p'_i)} \\
&= \exp\left(-t_i\left(\ln\left(\frac{1}{\alpha_i}\right) - \ln\left(\frac{e t'_{i-1}}{t'_i}\right)\right)\right) + \Delta e^{-\Omega(\delta^2 p'_i)} \\
&\le \exp\left(-t'_i\left((1-o(1))\frac{p'_i}{K t'_{i-1}} - \ln\left(\frac{e t'_{i-1}}{t'_i}\right)\right)\right) + \Delta e^{-\Omega(\delta^2 p'_i)} && \ln\frac{1}{\alpha_i} = (1-o(1))\frac{p'_i}{K t'_{i-1}} \\
&\le \exp\left(-\left((1-o(1))\frac{T}{K} - t'_i\ln(e\Delta)\right)\right) + \Delta e^{-\Omega(\delta^2 p'_i)} && \text{defn. } t'_i \text{ and } t'_{i-1}/t'_i < \Delta \\
&\le \exp\left(-T\left(\frac{(1-o(1))}{K} - \frac{t'_{i-1}}{p'_i}\ln(e\Delta)\right)\right) && + \Delta e^{-\Omega(\delta^2 p'_i)} \\
&\le \exp\left(-T\left((1-o(1))\frac{1}{K} - \frac{2\ln(e\Delta)}{\Delta^{0.1\gamma}}\right)\right) + \Delta e^{-\Omega(\delta^2 p'_i)} && \frac{t'_{i-1}}{p'_i} \le \frac{2T}{p'_i} \le \frac{2}{\Delta^{0.1\gamma}} \\
&\le \exp\left(-\Omega(T)\right) + \Delta e^{-\Omega(\delta^2 p'_i)}
\end{aligned}
$$

□

4.2.1 Union bound or constructive Lovász Local Lemma

We want to ensure that H_i holds for every round i. If H_{i-1} is true, then $\Pr(\overline{H}_i(e)) \le \exp\left(-\Delta^{1-0.95\gamma}\right)$ and $\Pr(\overline{H}_{i,c}(u)) \le \exp\left(-\Delta^{1-0.95\gamma}\right)$. If $\Delta^{1-\gamma} \ge \log n$, then each of the bad event occur with probability at most $1/\operatorname{poly}(n)$. Since there are at most $O(n^3)$ events, by the union bound, H_i holds w.h.p. On the other hand, if $\Delta^{1-\gamma} \le \log n$, then one can use the constructive Lovász Local Lemma (LLL) to make H_i hold w.h.p. Suppose that the probability each event happens is at most p and each event is dependent with at most d other events. If $ep(d+1) < 1$, the LLL guarantees that the probability none of the events happen is positive. In Chapter 2, we showed that if a stronger condition of LLL, $epd^2 < 1$, is satisfied, then the assignment can be constructed more efficiently, in $O(\log_{1/epd^2} n)$ rounds w.h.p.

Now, each of the bad events $\overline{H}_{i,c}(u)$ or $\overline{H}_i(e)$ is dependent with other events only if their distance is at most 3. (The distance between two edges is the distance in the line graph; the distance between a vertex and an edge is the distance between the vertex and the further endpoint of the edge). Since there are $O(\Delta)$ events on each vertex and $O(1)$ events on each edge, each event depends on at most $d = O(\Delta^3 \cdot \Delta) = O(\Delta^4)$ events. Let $p = \exp(-\Delta^{1-0.95\gamma})$ be an upper bound on the probability of each bad event. Now we have $epd^2 \leq \exp(-\Delta^{1-\gamma})$. Therefore, we can make H_i hold in $O(\log_{1/epd^2} n) \leq O(\log n/\Delta^{1-\gamma})$ rounds w.h.p. This completes the proof of Theorem 4.2.1.

Note that our proof for Theorem 4.2.1 does not rely on all the palettes being identical. Therefore, our algorithm works as long as each palette has at least $(1+\epsilon)\Delta$ colors, which is known as the list-edge-coloring problem.

Chapter 5

The $(\Delta+1)$-Coloring Problem

5.1 Introduction

The study of the $(\Delta+1)$-coloring problem can be traced back to the seminal works of Luby [120] and Alon, Babai and Itai [3], who devised $O(\log n)$-time algorithms for Maximal Independent Set problem. Luby [120] showed a reduction from the $(\Delta+1)$-coloring problem to MIS problem, so that the $(\Delta+1)$-coloring problem can be solved in $O(\log n)$ rounds.

Remarkably, even though these problems have been intensively investigated for the last three decades (see Section 4.1.1 for a short overview of some of the most related results), the logarithmic bound [3, 120] remains the state-of-the-art to this date. Indeed, the currently best-known algorithm for these problems (due to Barenboim et al. [11]) requires $O(\log\Delta)+\exp(O(\sqrt{\log\log n}))$ time. However, for $\Delta=n^{\Omega(1)}$ this bound is no better than the logarithmic bound of [3, 120].

In this chapter, we give a sublogarithmic algorithm for $(\Delta+1)$-vertex-coloring in $(1-\epsilon)$-locally sparse graphs. A graph $G=(V,E)$ is said to be $(1-\epsilon)$-*locally sparse* if for every vertex $v\in V$, its neighborhood $\Gamma(v)=\{u\mid(v,u)\in E\}$ induces at most $(1-\epsilon)\binom{\Delta}{2}$ edges. We devise a $(\Delta+1)$-vertex-coloring algorithm for $(1-\epsilon)$-locally sparse graphs that runs in $O(\log^*\Delta+\log 1/\epsilon)$ rounds for any $\epsilon>0$, provided that $\epsilon\Delta=(\log n)^{1+\Omega(1)}$. Without this restriction on the range of Δ our algorithm has running time $O(\log(1/\epsilon))+\exp(O(\sqrt{\log\log n}))$.

Our result shows that the only "hurdle" that stands on our way towards a sublogarithmic-time $(\Delta+1)$-vertex-coloring algorithm is the case of dense graphs. In particular, these graphs

must have arboricity[1] $\lambda(G) > (1-\epsilon)\Delta/2$, for any constant $\epsilon > 0$. (Note that $\lambda(G) \leq \Delta/2$.) Remarkably, graphs with arboricity close to the maximum degree are already known to be the only hurdle that stands on the way towards devising a deterministic polylogarithmic-time $(\Delta+1)$-vertex-coloring algorithm. Specifically, Barenboim and Elkin [8] devised a deterministic polylogarithmic-time algorithm that $(\Delta+1)$-vertex-colors all graphs with $\lambda(G) \leq \Delta^{1-\epsilon}$, for some constant $\epsilon > 0$.

Moreover, this result also implies that $(2\Delta-1)$-edge coloring can be solved in $e^{O(\sqrt{\log\log n})}$ rounds. It is easy to see that in a line graph of degree $\Delta = 2(\Delta'-1)$ (Δ' is the degree of its underlying graph) every neighborhood induces at most $(\Delta'-1)^2 = (\Delta/2)^2 = (1/2+1/2(\Delta-1))\binom{\Delta}{2}$ edges. Hence, our $(\Delta+1)$-vertex-coloring algorithm requires only $\exp(O(\sqrt{\log\log n}))$ time for $\Delta' \geq 2$. (For $\Delta' = O(1)$ a graph can be $(2\Delta'-1)$-edge-colored in $O(\Delta' + \log^* n) = O(\log^* n)$ time, using a classical $(2\Delta'-1)$-edge-coloring algorithm of Panconesi and Rizzi [140].)

The notion of $(1-\epsilon)$-locally sparse graphs was introduced by Alon, Krivelevich and Sudakov [4] and was studied also by Vu [172]. Distributed vertex-coloring of sparse graphs was studied in numerous papers. See, e.g., [7, 10, 11, 21, 149, 157], and the references therein.

Technical Overview We use a twofold approach. We will first analyze just *one* round of the standard trial algorithm, where each vertex randomly selects exactly one color from its palette. We show that because the neighborhood is sparse, at least $\Omega(\epsilon\Delta)$ pairs of neighbors will be assigned the same color, and so the palette size will concentrate at a value $\Omega(\epsilon\Delta)$ larger than its degree. Then by using the idea of selecting multiple colors, we develop an algorithm that colors the graph rapidly. In this algorithm, instead of selecting the colors with a uniform probability as in the edge coloring algorithm, vertices may select different probabilities that are inversely proportional to their palette sizes. Note that Schneider and Wattenhofer [157] showed that $(1+\epsilon)\Delta$-vertex coloring problem can be solved in $O(\log(1/\epsilon) + \log^* n)$ rounds if $\Delta \gg \log n$. However, it is not clear whether their proof extends directly to the case where palettes can be non-uniform as in our case.

[1]The arboricity $\lambda(G)$ of a graph G is the minimum number of edge-disjoint forests required to cover the edge set of G.

5.2 Coloring $(1-\epsilon)$-Locally Sparse Graphs with $\Delta+1$ colors

In this section and the following section we switch contexts from edge coloring to vertex coloring. Now the palette after round i, $P_i(u)$, is defined on the vertices rather than on the edges. G_i is the graph obtained by deleting those already colored vertices. Also, we assume each vertex has an unique ID, $\mathrm{ID}(u)$. Redefine the set functions $N_i(u) : V \to 2^V$, $N_{i,c}(u) : V \to 2^V$, $N^*_{i,c}(u) : V \to 2^V$ to be the neighboring vertices of u, the neighboring vertices of u having c in their palettes, and the neighboring vertices of u having smaller ID than u and having c in their palette.

G is said to be $(1-\epsilon)$-locally sparse if for any $u \in G$, the number of edges spanning the neighborhood of u is at most $(1-\epsilon)\binom{\Delta}{2}$ (i.e. $|\{xy \in G \mid x \in N(u) \text{ and } y \in N(u)\}| \leq (1-\epsilon)\binom{\Delta}{2}$).

Theorem 5.2.1. *Let $\epsilon, \gamma > 0$ and G be a $(1-\epsilon)$-locally sparse graph. There exists a distributed algorithm that colors G with $\Delta+1$ colors in $O(\log^* \Delta + \log(1/\epsilon) + 1/\gamma)$ rounds if $(\epsilon\Delta)^{1-\gamma} = \Omega(\log n)$.*

Corollary 5.2.2. *Let $\epsilon > 0$ and G be a $(1-\epsilon)$-locally sparse graph. G can be properly colored with $(\Delta+1)$ colors in $O(\log(1/\epsilon) + e^{O(\sqrt{\log\log n})})$ rounds.*

Proof. Let $\gamma = 1/2$. If $\epsilon\Delta = \Omega(\log^2 n)$, Theorem 5.2.1 gives an algorithm that runs in $O(\log^* \Delta + \log(1/\epsilon))$ rounds. Otherwise if $\epsilon\Delta = O(\log^2 n)$, the $(\Delta+1)$-coloring algorithm given in [11] runs in $O(\log \Delta + e^{O(\sqrt{\log\log n})}) = O(\log \frac{\log n}{\epsilon} + e^{O(\sqrt{\log\log n})}) = O(\log(1/\epsilon) + e^{O(\sqrt{\log\log n})})$ rounds. □

First we assume that each vertex $u \in G$ has Δ neighbors. If a vertex u has less than Δ neighbors, we will attach $\Delta - \deg(u)$ imaginary neighbors to it. We will analyze the following process for just a single round. Initially every vertex has palette $P_0(u) = \{1, \ldots \Delta+1\}$. Each vertex picks a tentative color uniformly at random. For each vertex, if no neighbors of smaller ID picked the same color, then it will color itself with the chosen color. Now each vertex removes the colors that are colored by its neighbors. Let $\deg_1(u)$ and $P_1(u)$ denote the degree of u and the palette of u after the first round. The idea is to show

that $|P_1(u)| \geq \deg_1(u) + \Omega(\epsilon\Delta)$, then we can apply the algorithm in the previous section. Intuitively this will be true, because of those neighbors of u who become colored, some fraction of them are going to be colored the same, since the neighborhood of u is not entirely spanned.

Let $N(u)$ denote u's neighbors. For $x, y \in N(u)$, we call xy a non-edge if $xy \notin E$. For $x, y \in N(u)$ where $\mathrm{ID}(x) < \mathrm{ID}(y)$, we call xy a *successful non-edge* w.r.t. u if the following two condition holds: First, xy is not an edge and x and y are colored with the same color. Second, aside from x, y, no other vertices in $N(u)$ with smaller ID than y picked the same color with x, y. We will show that w.h.p. there will be at least $\epsilon\Delta/(8e^3)$ successful non-edges. Then $|P_1(u)| \geq \Delta + 1 - (\Delta - \deg_1(u)) + \epsilon\Delta/(8e^3) \geq \deg_1(u) + \epsilon\Delta/(8e^3)$.

Lemma 5.2.3. *Fix a vertex $u \in G$. Let Z denote the number of successful non-edges w.r.t. u.*

$$\Pr(Z < \epsilon\Delta/(8e^3)) \leq e^{-\Omega(\epsilon\Delta)}$$

Proof. We will assume without loss of generality that the neighborhood of u has exactly $(1-\epsilon)\binom{\Delta}{2}$ edges. This can be assumed without loss of generality, because we can arbitrarily add edges to its neighborhood until there are $(1-\epsilon)\binom{\Delta}{2}$ edges. If Z' is the number of successful non-edges in the modified scenario, then Z statistically dominates Z', i.e. $\Pr(Z \geq z) > \Pr(Z' \geq z)$. Given the same outcomes of the random variables, if a pair xy is a successful non-edge in the modified scenario, then it must also be a successful non-edge in the original scenario.

We will first show that the expected number of successful non-edges is at least $\epsilon\Delta/(4e^3)$. Then we will define a martingale sequence on the 2-neighborhood of u. After showing the variance $\sum_i V_i$ has the same order as its expectation, $O(\epsilon\Delta)$, we will apply the method of bounded variance (Lemma A.8) to get the stated bound.

Given a non-edge xy in the neighborhood of u, the probability it is successful is at least $(1 - 1/(\Delta+1))^{3\Delta-2} \cdot (1/(\Delta+1)) = (1 - 1/(\Delta+1))^{3\Delta-1} \cdot (1/\Delta) \geq e^{-3}/\Delta$. The expectation

(assuming $\Delta > 1$)

$$\begin{aligned}\mathrm{E}[Z] &= \sum_{\substack{xy \notin E \\ x,y \in N(u)}} \Pr(xy \text{ is successful}) \\ &\geq \frac{\epsilon\Delta(\Delta-1)}{2} \cdot \frac{e^{-3}}{\Delta} = \frac{\epsilon(\Delta-1)}{2e^{-3}} \geq \frac{\epsilon\Delta}{4e^{-3}}\end{aligned}$$

We will define the martingale sequence on the 2-neighborhood of u and then show the variance $\sum_i V_i$ has the same order with its expectation, $O(\epsilon\Delta)$. Let $\{u_0 = u, u_1, \ldots u_k\}$ be the vertices in the 2-neighborhood of u, where vertices with distance 2 are listed first and then distance 1. The distance 1 vertices are listed by their ID in increasing order. Let X_i denote the color picked by u_i. Given $\boldsymbol{X}_{i-1}$, let D_{i,s_i} be $|\mathrm{E}[Z \mid \boldsymbol{X}_{i-1}, X_i = s_i] - \mathrm{E}[Z \mid \boldsymbol{X}_{i-1}]|$ and V_i be $\mathrm{Var}(\mathrm{E}[Z \mid \boldsymbol{X}_i] - \mathrm{E}[Z \mid \boldsymbol{X}_{i-1}] \mid \boldsymbol{X}_{i-1})$. Note that (see [40])

$$\sqrt{V_i} \leq \max_{s_i} D_{i,s_i} \leq \max_{s_i, s_i'} |\mathrm{E}[Z \mid \boldsymbol{X}_{i-1}, X_i = s_i] - \mathrm{E}[Z \mid \boldsymbol{X}_{i-1}, X_i = s_i']|$$

Also, $\mathrm{E}[Z \mid \boldsymbol{X}_i] = \sum_{x,y \in N(u), xy \notin E} \mathrm{E}[xy \text{ is successful} \mid \boldsymbol{X}_i]$. We discuss the cases whether u_i is a neighbor of u separately. If $u_i \notin N(u)$, whether u_i chose s_i or s_i' only affects on those non-edges xy such that at least one of x or y is adjacent to u_i. Let E_i denote such a set of non-edges. If $xy \in E_i$, then

$$|\mathrm{E}[xy \text{ is successful} \mid \boldsymbol{X}_{i-1}, X_i = s_i] - \mathrm{E}[xy \text{ is successful} \mid \boldsymbol{X}_{i-1}, X_i = s_i']| \leq 2/(\Delta+1)^2$$

because they only differ when both x and y picked s_i or s_i'. Thus, $\max_{s_i} D_{i,s_i} \leq 2|E_i|/(\Delta+1)^2$. Notice that $|E_i| \leq \epsilon\Delta^2$ and $\sum_i |E_i| \leq \epsilon\Delta^2 \cdot (2\Delta) \leq 2\epsilon\Delta^3$, since each of two endpoints of a non-edge can be incident to $\epsilon\Delta^2$ edges in those E_i. This implies $\sum_i |E_i|^2 \leq 2\epsilon^2\Delta^5$, since the sum is maximized when each $|E_i|$ is either 0 or $\epsilon\Delta^2$. Therefore, $\sum_{i:u_i \in N(N(u)) \setminus N(u)} V_i \leq \sum_i 4|E_i|^2/(\Delta+1)^4 \leq 8\epsilon^2\Delta$.

On the other hand, if $u_i \in N(u)$, we will first bound $D_{i,s_i} = |\mathrm{E}[Z \mid \boldsymbol{X}_i] - \mathrm{E}[Z \mid \boldsymbol{X}_{i-1}]|$ for a fixed s_i. Then we will bound $V_i = \sum_{s_i} \Pr(X_i = s_i) \cdot D_{i,s_i}^2$. Again, we break Z into sum of random variables $\sum_{u_a u_b \notin E, u_a, u_b \in N(u)} X_{u_a u_b}$, where $X_{u_a u_b}$ is the event that the non-edge $u_a u_b$ is successful. The indices a, b are consistent with our martingale sequences. Without loss of generality, we assume $a < b$ and so $\mathrm{ID}(u_a) < \mathrm{ID}(u_b)$. Let

$D_{i,s_i,ab} = |\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i] - \mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}]|$. In order to derive an upper bound for $(\sum D_{i,s_i,ab})^2$, we divide the non-edges u_a, u_b into five cases.

1. $a < b < i$: In this case, the color chosen by u_i does not affect $\mathrm{E}[X_{u_a u_b}]$, because u_i has a higher ID. Thus, $D_{i,s_i,ab} = 0$.

2. $i < a < b$: In this case,

$$D_{i,s_i,ab} \leq |\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i] - \mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i']| \leq 2/(\Delta+1)^2$$

   because they only differ when u_a and u_b both picked s_i or s_i'. There are at most $\epsilon\Delta^2$ edges affected. Therefore, $\sum_{i<a<b} D_{i,s_i,ab} \leq 2\epsilon$.

3. $a < i < b$: If $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}] = 0$, then $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i] = 0$, which creates no difference. If $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}]$ is not zero, then it is the case that u_a has picked its color uniquely among $(N(u) \cap \{u_1, \ldots, u_{i-1}\}) \cup N^*(u_a)$. Therefore, $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}] = (1 - 1/(\Delta+1))^{b-i} \cdot 1/(\Delta+1)$. If u_a chose s_i, then $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i] = 0$. Otherwise, $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i] = (1 - 1/(\Delta+1))^{b-i-1} \cdot 1/(\Delta+1)$. In the former case, the difference is at most $1/(\Delta+1)$. In the latter case, the difference is at most $(1 - 1/(\Delta+1))^{b-i-1} \cdot 1/(\Delta+1) - (1 - 1/(\Delta+1))^{b-i} \cdot 1/(\Delta+1) \leq 1/(\Delta+1)^2$. Notice that among the non-edges $u_a u_b$ with $a < i < b$, only those with u_a uniquely colored s_i among $(N(u) \cap \{u_1, \ldots, u_{i-1}\})$ fits into the former case. Denote the edge set by E_{s_i}, we have $\sum_{a<i<b} D_{i,s_i,ab} \leq \epsilon + |E_{s_i}|/(\Delta+1)$. Also note that $\sum_{s_i} |E_{s_i}| \leq \epsilon\Delta^2$, since E_{s_i} is disjoint from $E_{s_i'}$ if $s_i \neq s_i'$.

4. $a = i < b$: In this case,

$$D_{i,s_i,ab} \leq |\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i] - \mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i']| \leq 2/(\Delta+1)$$

   because they are different only when u_b picked s_i or s_i'. There are at most $\overline{\deg}(u_i) \stackrel{\text{def}}{=} \Delta - \deg(u_i)$ non-edges affected. Therefore, $\sum_{a=i<b} D_{i,s_i,ab} \leq \overline{\deg}(u_i)/(\Delta+1)$.

5. $a < i = b$: In this case, $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}, X_i = s_i]$ is either 1 or 0. Note that $\mathrm{E}[X_{u_a u_b} \mid \boldsymbol{X}_{i-1}]$ is at most $1/(\Delta+1)$. Therefore, if s_i is the color picked by u_a and u_a is the only vertex that picked s_i among $u_1 \ldots, u_{i-1}$, then $D_{i,s_i,ab}$ is at most 1. Otherwise,

it is at most $1/(\Delta+1)$. Let μ_{s_i} be the indicator variables whether there exists such a u_a that colored s_i. We have $\sum_{a<i=b} D_{i,s_i,ab} \leq \mu_{s_i} + \overline{\deg}(u_i)/(\Delta+1)$. Note that $\sum_{s_i} \mu_{s_i} \leq \overline{\deg}(u_i)$.

Now we are ready to bound the variance V_i. For readability we let $\Delta_1 = \Delta + 1$.

$$\begin{aligned}
V_i &= \sum_{s_i} \Pr(X_i = s_i) \cdot D_i^2 \\
&\leq \sum_{s_i} \frac{1}{\Delta_1} \cdot \left(\sum_{a<b<i} D_{i,s_i,ab} + \sum_{i<a<b} D_{i,s_i,ab} + \sum_{a<i<b} D_{i,s_i,ab} + \sum_{a=i<b} D_{i,s_i,ab} + \sum_{a<b=i} D_{i,s_i,ab} \right)^2 \\
&\leq \sum_{s_i} \frac{1}{\Delta_1} \cdot \left(3\epsilon + \frac{|E_{s_i}|}{\Delta_1} + \frac{2\overline{\deg}(u_i)}{\Delta_1} + \mu_{s_i} \right)^2 \\
&\leq \frac{7}{\Delta_1} \cdot \sum_{s_i} \left((3\epsilon)^2 + \left(\frac{|E_{s_i}|}{\Delta_1} \right)^2 + \left(\frac{2\overline{\deg}(u_i)}{\Delta_1} \right)^2 + \mu_{s_i}^2 \right)
\end{aligned}$$

The last inquality follows since $(x_1+x_2+x_3+x_4)^2 \leq 7(x_1^2+x_2^2+x_3^2+x_4^2)$. Note that $\sum_{s_i}(3\epsilon)^2 \leq 9\Delta_1\epsilon^2$, $\sum_{s_i}\left(\frac{|E_{s_i}|}{\Delta_1}\right)^2 \leq \epsilon\Delta$, $\sum_{s_i}\left(\frac{2\overline{\deg}(u_i)}{\Delta_1}\right)^2 \leq \frac{4\overline{\deg}(u_i)^2}{\Delta_1}$, and $\sum_{s_i}\mu_{s_i}^2 \leq \overline{\deg}(u_i)$. Therefore,

$$V_i \leq \frac{7}{\Delta_1}\left(9\Delta_1\epsilon^2 + \epsilon\Delta + \frac{4\overline{\deg}(u_i)^2}{\Delta_1} + \overline{\deg}(u_i) \right)$$

Now notice that $\sum_i \overline{\deg}(u_i) \leq \epsilon\Delta^2$ and $\sum_i \overline{\deg}^2(u_i)$ is a sum of convex functions, which is maximized when each term is either 0 or the maximum. Therefore, $\sum_i \overline{\deg}^2(u_i) \leq \epsilon\Delta^3$. We have

$$\sum_{i:u_i \in N(u)} V_i \leq 7(9\epsilon\Delta + \epsilon\Delta + 4\epsilon\Delta + \epsilon\Delta) \leq 105\epsilon\Delta$$

In order to apply Lemma A.8, we have to bound $\max_{s_i} D_{i,s_i}$. Notice that for any two outcome vectors $\boldsymbol{X}, \boldsymbol{X}'$ that only differ at the i'th coordinate, Z differs by at most 2. That is, by changing the color of a vertex $x \in N(u)$ from s_i to s_i', the number of successful non-edges can only differ by 2. First, this is true if $x = u$ or x is at distance 2 from u, since it can only create at most one sucessful edge when x unselects s_i and destroy one when x selects s_i'. When $x \in N(u)$, we consider the effect when x unselects the color s_i. It can create or

destroy at most 1 successful non-edge. It creates a successful non-edge yz only when x, y, z picked s_i and no other vertices in $N(u)$ with smaller ID than y, z picked s_i. It destroys a non-edge when xy was a successful non-edge that both colored s_i. Note that if such a y exists, there can be at most one, by the definition of successful non-edge. Similarly, it can create or destroy at most 1 successful non-edge when x picks s'_i. It can be shown that this 2-Lipschitz condition implies $D_{i,s_i} \leq 2$ [40, Corollary 5.2].

Applying A.8 with $t = \epsilon\Delta/(8e^3)$ and $M = 2$, we get that

$$\begin{aligned}
&\Pr(Z < \epsilon\Delta/(8e^3)) = \Pr(Z < \epsilon\Delta/(4e^3) - t) \\
&\leq \exp\left(-\frac{t^2}{2(105\epsilon\Delta + 8\epsilon^2\Delta + 2t/3)}\right) \\
&= \exp(-\Omega(\epsilon\Delta)).\square
\end{aligned}$$

Therefore, by Lemma 5.2.3, for any $u \in G$,

$$\Pr\left(|P_1(u)| < \deg_1(u) + \frac{\epsilon}{8e^3} \cdot \Delta\right) \leq e^{-\Omega(\epsilon\Delta)}$$

If $\epsilon\Delta = \Omega(\log n)$, then $\Pr(|P_1(u)| < \deg_1(u) + \frac{\epsilon}{8e^3} \cdot \Delta) \leq e^{-\Omega(\epsilon\Delta)} \leq 1/\operatorname{poly}(n)$. By the union bound, $|P_1(u)| \geq \deg_1(u) + \frac{\epsilon}{8e^3} \cdot \Delta)$ holds for all $u \in G$ with high probability. If $(\epsilon\Delta)^{1-\gamma} = \Omega(\log n)$, we show the rest of the graph can be colored in $O(\log^* \Delta + \log(1/\epsilon) + 1/\gamma)$ rounds in the next section.

5.3 Vertex Coloring with $\deg(u) + \epsilon\Delta$ Colors

In this section we consider the vertex coloring problem where each vertex has $\epsilon\Delta$ more colors in its palette than its degree. The goal is to color each vertex by using a color from its palette. Note that the palette of each vertex may not necessarily be identical and can have different sizes.

Theorem 5.3.1. *Given ϵ, $\gamma > 0$, and G, where each vertex $u \in G$ has a palette containing at least $\deg(u) + \epsilon\Delta$ colors and $(\epsilon\Delta)^{1-\gamma} = \Omega(\log n)$. There exists a distributed algorithm that colors G properly in $O(\log^* \Delta + 1/\gamma + \log(1/\epsilon))$ rounds.*

Corollary 5.3.2. *Suppose that each vertex $u \in G$ has a palette containing at least* $\deg(u)+\epsilon\Delta$ *colors, then G can be properly colored in* $O(\log(1/\epsilon) + e^{O(\sqrt{\log\log n})})$ *rounds.*

Proof. Let $\gamma = 1/2$. If $\epsilon\Delta = \Omega(\log^2 n)$, Theorem 5.3.1 gives an algorithm that runs in $O(\log^* \Delta + \log(1/\epsilon))$ rounds. Otherwise if $\epsilon\Delta = O(\log^2 n)$, the $(\Delta+1)$-coloring algorithm given in [11] runs in $O(\log \Delta + e^{O(\sqrt{\log\log n})}) = O(\log \frac{\log n}{\epsilon} + e^{O(\sqrt{\log\log n})}) = O(\log(1/\epsilon) + e^{O(\sqrt{\log\log n})})$ rounds. □

We will define d_i in Algorithm 11 later. Algorithm 11 is modified from Algorithm 9. The first modification is that instead of running it on the edges, we run it on vertices. Second, instead of removing all colors picked by the neighbors from the palette, we only removes colors that are actually colored by their neighbors. Third, instead of selecting colors with identical probabilty for each vertex, the vertices may select with different probabilities.

Vertex-Coloring-Algorithm($G, \{d_i\}$)
1: $G_0 \leftarrow G$
2: $i \leftarrow 0$
3: **repeat**
4: $\quad i \leftarrow i+1$
5: $\quad$ **for** each $u \in G_{i-1}$ **do**
6: $\quad\quad$ Include each $c \in P_{i-1}(u)$ in $S_i(e)$ independently with probability
7: $\quad\quad\quad \pi_i(u) = \frac{1}{|P_{i-1}(u)|} \cdot \frac{d_{i-1}+\epsilon\Delta}{d_{i-1}+1}$.
8: $\quad\quad$ If $S_i(u) \setminus S_i(N^*_{i-1}(u)) \neq \emptyset$, u color itself with any color in $S_i(u) \setminus S_i(N^*_{i-1}(u))$.
9: $\quad\quad$ Set $P_i(u) \leftarrow P_{i-1}(u) \setminus \{c \mid$ a neighbor of u is colored $c\}$.
10: $\quad$ **end for**
11: $\quad G_i \leftarrow G_{i-1} \setminus \{$colored vertices$\}$
12: **until**

Algorithm 11

Due to the second modification, at any round of the algorithm, a vertex always has $\epsilon\Delta$ more colors in its palette than its degree. The intuition of the third modification is that if every vertex selects with an identical probability, then a neighbor of u having a palette with very large size might prevent u to become colored. To avoid this, the neighbor of u should choose

each color with a lower probability. Define the parameters as follows:

$$d_0 = \Delta \qquad T = (\epsilon\Delta)^{1-\gamma} \qquad \alpha_i = e^{-\frac{d_{i-1}+\epsilon\Delta}{8(d_{i-1}+1)}}$$
$$d_i = \begin{cases} \max(1.01\alpha_i d_{i-1}, T) & \text{if } d_{i-1} > T \\ \frac{T}{\epsilon\Delta} \cdot d_{i-1} & \text{otherwise} \end{cases}$$

Let $H_i(u)$ denote the event that $\deg_i(u) \leq d_i$ after round i. Let H_i denote the event that $H_i(u)$ holds for all $u \in G_{i-1}$, where G_{i-1} is the graph induced by the uncolored vertices after round $i-1$. Note that when H_{i-1} is true,

$$\pi_i(u) = \frac{1}{|P_{i-1}(u)|} \cdot \frac{d_{i-1}+\epsilon\Delta}{d_{i-1}+1} \leq \frac{1}{|P_{i-1}(u)|} \cdot \frac{\deg_{i-1}(u)+\epsilon\Delta}{\deg_{i-1}(u)+1} \leq \frac{1}{\deg_{i-1}(u)+1}$$

Notice that u remains uncolored iff it did not select any color in $P_{i-1}(u) \setminus S_i(N^*_{i-1}(u))$. We will show that the size of $P_{i-1}(u) \setminus S_i(N^*_{i-1}(u))$ is at least $|P_{i-1}(u)|/8$ and so the probability u did not become colored is at most $(1-\pi_i(u))^{|P_{i-1}(u)|/8} \leq \alpha_i$. Then, the expected value of $\deg_i(u)$ will be at most $\alpha_i d_{i-1}$. Depending on whether $d_{i-1} > T$, we separate the definition of d_i into two cases, because we would like the tail probability that d_i deviates from its expectation to be bounded by $e^{-\Omega(T)}$.

Lemma 5.3.3. *$d_i < 1$ for some $i = O(\log^* \Delta + 1/\gamma + \log(1/\epsilon))$.*

Proof. We analyze how d_i decreases in three stages. The first stage is when $d_{i-1} > \epsilon\Delta/33$. During this stage,

$$\begin{aligned} d_i &= 1.01\alpha_i d_{i-1} \\ &\leq 1.01 \exp\left(-\frac{d_{i-1}+\epsilon\Delta}{8(d_{i-1}+1)}\right) \cdot d_{i-1} \\ &\leq 1.01 \exp\left(-1/16\right) \cdot d_{i-1} && d_{i-1} \geq 1 \\ &\leq 0.99 \cdot d_{i-1} \end{aligned}$$

Therefore, this stage ends in $O(\log(1/\epsilon))$ rounds. The second stage starts at first r_1 such

that $T < d_{r_1-1} \le \epsilon\Delta/33$. When $i > r_1$:

$$\begin{aligned}
\alpha_i &\le 1.01 \cdot \exp\left(-\frac{d_{i-1} + \epsilon\Delta}{16 d_{i-1}}\right) \\
&\le 1.01 \cdot \exp\left(-\frac{\epsilon\Delta}{16 d_{i-1}}\right) \\
&\le \exp\left(\frac{1}{32}\right) \cdot \exp\left(-\frac{\epsilon\Delta}{16 d_{i-1}}\right) \\
&\le \exp\left(-\frac{\epsilon\Delta}{32 d_{i-1}}\right) && d_{i-1} \le \epsilon\Delta/33 \le \epsilon\Delta \\
&\le \exp\left(-\frac{\epsilon\Delta}{33 \alpha_{i-1} d_{i-2}}\right) \\
&\le \exp\left(-\frac{1}{\alpha_{i-1}}\right) && d_{i-2} \le \epsilon\Delta/33
\end{aligned}$$

Therefore, $\frac{1}{\alpha_{r_1+\log^*(1.01\Delta)+1}} \ge \underbrace{e^{e^{\cdot^{\cdot^{e}}}}}_{\log^*(1.01\Delta)} \ge 1.01\Delta$, and so $d_{r_1+\log^*(1.01\Delta)+1} \le \max(1.01\alpha_{r_1+\log^*(1.01\Delta)+1}\Delta, T) \le T$.

The third stages begins at the first round r_2 such that $d_{r_2-1} = T$. If $i \ge r_2$, then $d_i = \frac{T}{\epsilon\Delta} \cdot d_{i-1} \le (\epsilon\Delta)^{-\gamma} \cdot d_{i-1}$. Therefore, $d_{r_2+1/\gamma+1} < (\epsilon\Delta)^{-1} \cdot T < 1$. The total number of rounds is $O(\log(1/\epsilon) + \log^* \Delta + 1/\gamma)$. □

Lemma 5.3.4. *Suppose that H_{i-1} holds, then* $\Pr(\deg_i(u) > d_i) \le e^{-\Omega(T)} + \Delta e^{-\Omega(\epsilon\Delta)}$.

Proof. Let $\widehat{P}_i(x) \stackrel{\text{def}}{=} P_{i-1}(x) \setminus S_i(N^*_{i-1}(x))$ denote the current palette of x excluding the colors chosen by its neighbors. We will first show that $\mathrm{E}[|\widehat{P}_i(x)|] \ge |P_{i-1}(x)|/4$. Define $w(c) = \sum_{y \in N^*_{i-1,c}(x)} \pi_i(y)$. We defined $w(c)$ to simplify the calculation because we will argue that when $\sum_{c \in P_{i-1}(x)} w(c)$ is fixed, some inequality is minimized when each of the summand equals to $\sum_{c \in P_{i-1}(x)} w(c)/|P_{i-1}(x)|$. The probability c is not chosen by any of x's neighbors with smaller ID is

$$\prod_{y \in N^*_{i-1,c}(x)} (1 - \pi_i(y)) \ge \min_{\pi'_i : (\sum_{y \in N^*_{i-1,c}(x)} \pi'_i(y)) = w(c)} \prod_{y \in N^*_{i-1,c}(x)} (1 - \pi'_i(y))$$

which is minimized when $\pi'_i(y) = w(c)/\deg^*_{i-1,c}(u)$, so the quantity above is

$$\begin{aligned}
&\geq \left(1 - \frac{w(c)}{\deg^*_{i-1,c}(x)}\right)^{\deg^*_{i-1,c}(x)} \\
&= \left(1 - \frac{w(c)}{\deg^*_{i-1,c}(x)}\right)^{\frac{\deg^*_{i-1,c}(x)}{w(c)} \cdot w(c)} \\
&\geq \left(\frac{1}{4}\right)^{w(c)} && \frac{w(c)}{\deg^*_{i-1,c}(x)} \leq \frac{1}{2}
\end{aligned}$$

Note that the reason that $\frac{w(c)}{\deg^*_{i-1,c}(x)} \leq \frac{1}{2}$ is $\pi_i(y) \leq \frac{1}{\deg_{i-1}(y)+1} \leq \frac{1}{2}$ for $y \in N^*_{i-1,c}(x)$. Therefore,

$$\begin{aligned}
\mathrm{E}[|\widehat{P}_i(x)|] &= \sum_{c \in P_{i-1}(x)} \Pr(c \notin S_i(N^*_{i-1}(x))) \\
&\geq \sum_{c \in P_{i-1}(x)} \left(\frac{1}{4}\right)^{w(c)} \\
&\geq \min_{w' : \sum w(c) = \sum w'(c)} \sum_{c \in P_{i-1}(x)} \left(\frac{1}{4}\right)^{w'(c)}
\end{aligned}$$

which is minimized when $w'(c)$ are all equal, that is, $w'(c) = \sum_{c' \in P_{i-1}(x)} w(c')/|P_{i-1}(x)|$, hence

$$\geq |P_{i-1}(x)| \cdot \left(\frac{1}{4}\right)^{\sum_{c \in P_{i-1}(x)} w(c)/|P_{i-1}(x)|}$$

We show the exponent is at most 1, so that $\mathrm{E}[|\widehat{P}_i(x)|] \geq |P_{i-1}(x)|/4$. The exponent

$$\begin{aligned}
&\sum_{c \in P_{i-1}(x)} w(c)/|P_{i-1}(x)| = \\
&\sum_{y \in N^*_{i-1}(x)} \frac{|P_{i-1}(x) \cap P_{i-1}(y)|}{|P_{i-1}(y)|} \cdot \frac{d_{i-1} + \epsilon\Delta}{d_{i-1}+1} \cdot \frac{1}{|P_{i-1}(x)|} \\
&\leq \sum_{y \in N_{i-1}(x)} \frac{d_{i-1} + \epsilon\Delta}{d_{i-1}+1} \cdot \frac{1}{|P_{i-1}(x)|} \\
&\leq \frac{d_{i-1} + \epsilon\Delta}{d_{i-1}+1} \cdot \frac{\deg_{i-1}(x)}{|P_{i-1}(x)|} \\
&\leq 1 && \frac{\deg_{i-1}(x)}{|P_{i-1}(x)|} \leq \frac{d_{i-1}}{d_{i-1}+\epsilon\Delta}
\end{aligned}$$

Notice that the event whether the color $c \in S_i(N^*_{i-1}(x))$ is independent of other colors, so by a Chernoff Bound:

$$\begin{aligned}
\Pr(|\widehat{P}_i(x)| < |P_i(x)|/8) &\leq e^{-\Omega(|P_{i-1}(x)|)} \\
&= e^{-\Omega(\epsilon\Delta)}.
\end{aligned}$$

Let $x_1 \ldots x_k \in N_{i-1}(u)$ be the neighbors of u, listed by their ID in increasing order. Let $\mathcal{E}_j$ be the event that $|\widehat{P}_i(x_j)| \geq |P_i(x)|/8$ for all $x \in N_{i-1}(u)$. We have shown that $\Pr(\overline{\mathcal{E}}_j) \leq e^{-\Omega(\epsilon\Delta)}$. Let X_j denote x_j is not colored after this round. We will show that:

$$\max_{\boldsymbol{X}_{j-1}} \Pr(X_j \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \leq \alpha_i$$

Let $c' \in \widehat{P}_i(x_j)$. First we argue that $\Pr(c' \in S_i(x_j) \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) = \pi_i(u)$. Since $c' \in \widehat{P}_i(x_j)$, c' is not chosen by any of $x_1, \ldots, x_{j-1}$. Whether $X_1, \ldots, X_{j-1}$ hold does not depend on whether $c' \in S_i(x_j)$. Furthermore, the events $\mathcal{E}_1 \ldots \mathcal{E}_{j-1}$ do not depend on the colors chosen by x_j, since x_j has higher ID than $x_1, \ldots, x_{j-1}$. Also, $\mathcal{E}_j$ does not depend on

the colors chosen by x_j either. Therefore, $\Pr(X_j \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) = \pi_i(u)$ and we have:

$$\begin{aligned}
&\Pr(X_j \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \\
&= \prod_{c' \in P_i(e_j)} \Pr(c' \notin S_i(e_j) \mid \boldsymbol{X}_{j-1}, \mathcal{E}_1, \ldots, \mathcal{E}_j) \\
&\leq (1 - \pi_i(u))^{|P_{i-1}(u)|/8} && \mathcal{E}_j \text{ is true} \\
&\leq \exp\left(-\frac{d_{i-1} + \epsilon\Delta}{(d_{i-1}+1)|P_{i-1}(u)|} \cdot \frac{|P_{i-1}(u)|}{8}\right) && 1 - x \leq e^{-x} \\
&\leq \alpha_i
\end{aligned}$$

If $d_{i-1} > T$, by Lemma A.5 and Corollary A.2,

$$\begin{aligned}
&\Pr(\deg_i(u) > \max(1.01\alpha_i d_{i-1}, T)) \\
&\leq \Pr(\deg_i(u) > \max(1.01\alpha_i \deg_{i-1}(u), T)) \\
&\leq e^{-\Omega(T)} + \Delta e^{-\Omega(\Delta)}.
\end{aligned}$$

Otherwise we have $d_i = \frac{T}{\Delta\epsilon} \cdot d_{i-1} \leq (\epsilon\Delta)^{-\gamma} \cdot T$. By Lemma A.5 and Corollary A.2 with

$1 + \delta = T/(\alpha_i \epsilon \Delta)$,

$$
\begin{aligned}
&\Pr\left(\deg_i(u) > d_i\right) \\
&\leq \Pr\left(\deg_i(u) > \frac{T}{\alpha_i \epsilon \Delta} \cdot \alpha_i d_{i-1}\right) \\
&\leq \exp\left(-\alpha_i d_{i-1} \cdot \left(\frac{T}{\alpha_i \epsilon \Delta} \ln \frac{T}{\alpha_i \epsilon \Delta} - \left(\frac{T}{\alpha_i \epsilon \Delta} - 1\right)\right)\right) + \Delta e^{-\Omega(\epsilon\Delta)} \\
&\leq \exp\left(-d_i \left(\ln \frac{T}{e \alpha_i \epsilon \Delta}\right)\right) + \Delta e^{-\Omega(\epsilon\Delta)} \\
&\leq \exp\left(-d_i \left(\ln \frac{1}{\alpha_i} - \ln\left(\frac{e\epsilon\Delta}{T}\right)\right)\right) + \Delta e^{-\Omega(\epsilon\Delta)} \\
&\leq \exp\left(-d_i \left(\frac{\epsilon\Delta}{16 d_{i-1}} - \ln(e(\epsilon\Delta)^\gamma)\right)\right) + \Delta e^{-\Omega(\epsilon\Delta)} && \text{defn. } \alpha_i \\
&\leq \exp\left(-T \left(\frac{1}{16} - \frac{d_i}{T} \cdot \ln(e(\epsilon\Delta)^\gamma)\right)\right) + \Delta e^{-\Omega(\epsilon\Delta)} && \text{defn. } d_i \\
&\leq \exp\left(-T \left(\frac{1}{16} - \frac{1}{(\epsilon\Delta)^\gamma} \cdot \ln(e(\epsilon\Delta)^\gamma)\right)\right) + \Delta e^{-\Omega(\epsilon\Delta)} \\
&\leq \exp\left(-\Omega(T)\right) + \Delta e^{-\Omega(\epsilon\Delta)}
\end{aligned}
$$

In both cases, we have $\Pr(\deg_i(u) > d_{i+1}) \leq \exp(-\Omega(T)) + \Delta \exp(-\Omega(\epsilon\Delta))$ □

Since $(\epsilon\Delta)^{1-\gamma} = \Omega(\log n)$, $\Pr(\overline{H}_i(u)) \leq \exp(-\Omega(T)) + \Delta \exp(-\Omega(\epsilon\Delta)) \leq 1/\operatorname{poly}(n)$. By union bound H_i holds with high probability. After $O(\log^* \Delta + \log(1/\epsilon) + 1/\gamma)$ rounds, $\deg_i(u) = 0$ for all u w.h.p., and so the isolated vertices can color themselves with any colors in their palette.

Part II

Other Distributed Optimization Problems

Chapter 6

Almost-Tight Distributed Minimum Cut Algorithms

6.1 Introduction

The minimum cut is an important measure of networks. It determines, e.g., the network vulnerability and the limits to the speed at which information can be transmitted. While this problem has been well-studied in the centralized setting (e.g. [56, 94–97, 99, 121, 132, 160]), very little is known in the distributed setting, especially in the relevant context where communication links are constrained by a small *bandwidth* – the so-called CONGEST model (cf. Section 6.2).

Consider, for example, a simple variation of this problem, called λ-*edge-connectivity*: given an *unweighted* undirected graph G and a *constant* λ, we want to determine whether G is λ-edge-connected or not. In the centralized setting, this problem can be solved in $O(m + n\lambda^2 \log n)$ time [56], thus near-linear time when λ is a constant. (Throughout, n, m, and D denotes

the number of nodes, number of edges, and the network diameter, respectively.) In the distributed setting, however, non-trivial solutions are known only when $\lambda \leq 3$; this is due to algorithms of Pritchard and Thurimella [151] which can compute 2-edge-connected and 3-edge-connected components in $O(D)$ and $O(D + n^{1/2} \log^* n)$ time, respectively, with high probability[1]. This implies that the λ-edge-connectivity problem can be solved in $O(D)$ time when $\lambda = 2$ and $O(D + n^{1/2} \log^* n)$ time when $\lambda = 3$.

For the general version where input graphs could be weighted, the problem can be solved in near-linear time [95–97, 121] in the centralized setting. In the distributed setting, the first non-trivial upper bounds are due to Ghaffari and Kuhn [65], who presented $(2+\epsilon)$-approximation $O((\sqrt{n} \log^* n + D)\epsilon^{-5} \log^2 n \log\log n)$-time and $O(\epsilon^{-1})$-approximation $O(D + n^{\frac{1}{2}+\epsilon} \operatorname{poly}\log n)$-time algorithms. These upper bounds are complemented by a lower bound of $\Omega(D + n^{1/2}/\log n)$ for any approximation algorithm which was earlier proved by Das Sarma et al. [31] for the weighted case and later extended by [65] to the unweighted case. This means that the running times of the algorithms in [65] are tight up to a $\operatorname{polylog} n$ factor. Yet, it is still open whether we can achieve an approximation factor less than two in the same running time, or in fact, in any sublinear (i.e. $O(D + o(n))$) time.

Results. In this chapter, we present improved distributed algorithms for computing the minimum cut both exactly and approximately. Our exact deterministic algorithm for finding the minimum cut takes $O((\sqrt{n} \log^* n + D)\lambda^4 \log^2 n)$ time, where λ is the value of the minimum cut. Our approximation algorithm finds a $(1+\epsilon)$-approximate minimum cut in $O((D + \sqrt{n} \log^* n)\epsilon^{-5} \log^3 n)$ time with high probability. (If we only want to compute the $(1+\epsilon)$-approximate *value* of the minimum cut, then the running time can be slightly reduced to $O((\sqrt{n} \log^* n + D)\epsilon^{-5} \log^2 n \log\log n)$.) As noted earlier, prior to this work there was no sublinear-time exact algorithm even when λ is a constant greater than three, nor sublinear-time algorithm with approximation ratio less than two. Table 1 summarizes the results.

Techniques. The starting point of our algorithm is Thorup's tree packing theorem [165, Theorem 9], which shows that if we generate $\Theta(\lambda^7 \log^3 n)$ trees $T_1, T_2, \ldots$, where tree T_i is the

[1]We say that an event holds *with high probability* (w.h.p.) if it holds with probability at least $1 - 1/n^c$, where c is an arbitrarily large constant.

minimum spanning tree with respect to the loads induced by $\{T_1, \ldots, T_{i-1}\}$, then one of these trees will contain exactly one edge in some minimum cut (see Algorithm 6.4 for the definition of load). Since we can use the $O(\sqrt{n}\log^* n + D)$-time algorithm of Kutten and Peleg [114] to compute the minimum spanning tree (MST), the problem of finding a minimum cut is reduced to finding the minimum cut that 1-*respects a tree*; i.e., finding which edge in a given spanning tree defines a smallest cut (see the formal definition in Section 6.3). Solving this problem in $O(D + \sqrt{n}\log^* n)$ time is the first key technical contribution of this work. We do this by using a simple observation of Karger [97] which reduces the problem to computing the sum of degrees and the number of edges contained in a subtree rooted at each node. We use this observation along with Garay, Kutten and Peleg's *tree partitioning* [64, 114] to quickly compute these quantities. This requires several (elementary) steps, which we will discuss in more detail in Section 6.3.

The above result together with Thorup's tree packing theorem immediately imply that we can find a minimum cut exactly in $O((D + \sqrt{n}\log^* n)\lambda^7 \log^3 n)$ time. By using Karger's random sampling result [96] to bring λ down to $O(\log n/\epsilon^2)$, we can find a $(1+\epsilon)$-approximate minimum cut in $O((D + \sqrt{n}\log^* n)\epsilon^{-14}\log^{10} n)$ time. These time bounds unfortunately depend on large factors of λ, $\log n$ and $1/\epsilon$, which make their practicality dubious. Our second key technical contribution is a new algorithm which significantly reduces these factors by combining Thorup's greedy tree packing approach with Matula's contraction algorithm [121]. In Matula's $(2+\epsilon)$-approximation algorithm for the minimum cut problem, he partitioned the graph into *components* according to the *spanning forest decomposition* by Nagamochi and Ibaraki [132]. He showed that either a component induces a $(2+\epsilon)$-approximate minimum cut, or the minimum cut does not intersect with the components. In the latter case, it is safe to contract the components. Our algorithm uses a similar approach, but we partitions the graph according to Thorup's greedy tree packing approach instead of the spanning forest decomposition. We will show that either (i) a component induces a $(1+\epsilon)$-approximate minimum cut, (ii) the minimum cut does not intersect with the components, or (iii) the minimum cut 1-respect a tree in the tree packing. This algorithm and analysis will be discussed in detail in Algorithm 6.4. We note that our algorithm can also be implemented in the centralized setting in $O(m + n\epsilon^{-7}\log^3 n)$ time. It is slightly worse than the current best $O(m + n\epsilon^{-3}\log^3 n)$ by Karger [95].

Reference	Time	Approximation
Pritchard&Thurimella [151]	$O(D)$ for $\lambda \leq 2$	exact
Pritchard&Thurimella [151]	$O(\sqrt{n}\log^* n + D)$ for $\lambda \leq 3$	exact
This work	$O((\sqrt{n}\log^* n + D)\lambda^4 \log^2 n)$	exact
Das Sarma et al. [31]	$\Omega(\frac{\sqrt{n}}{\log n} + D)$	any
Ghaffari&Kuhn [65]	$O((\sqrt{n}\log^* n + D)\epsilon^{-5}\log^2 n \log\log n)$	$2+\epsilon$
This work	$O((\sqrt{n}\log^* n + D)\epsilon^{-5}\log^3 n)$	$1+\epsilon$

Table 1: Summary of results for minimum cut.

6.2 Preliminaries

Communication Model. We use a standard message passing network model called CONGEST [144]. A network of processors is modeled by an undirected unweighted n-node graph G, where nodes model the processors and edges model $O(\log n)$-*bandwidth* links between the processors. The processors (henceforth, nodes) are assumed to have unique IDs in the range of $\{1, \ldots, \text{poly}(n)\}$ and infinite computational power. We denote the ID of node v by $\text{ID}(v)$. Each node has limited topological knowledge; in particular, it only knows the IDs of its neighbors and knows *no* other topological information (e.g., whether its neighbors are linked by an edge or not). Additionally, we let $w : E(G) \to \{1, 2, \ldots, \text{poly}(n)\}$ be the edge weight assignment. The weight $w(uv)$ of each edge uv is known only to u and v. As is commonly done in the literature (e.g., [64, 65, 103, 114, 119, 133]), we will assume that the maximum weight is $\text{poly}(n)$ so that each edge weight can be sent through an edge (link) in one round.

There are several measures to analyze the performance of distributed algorithms. One fundamental measure is the *running time* defined as the worst-case number of *rounds* of distributed communication. At the beginning of each round, all nodes wake up simultaneously. Each node u then sends an arbitrary message of $B = \log n$ bits through each edge uv, and the message will arrive at node v at the end of the round. (See [144] for detail.) The running time is analyzed in terms of number of nodes and the diameter of the network, denoted by n and D respectively. Since we can compute n and 2-approximate D in $O(D)$ time, we will assume that every node knows n and the 2-approximate value of D.

Minimum Cut Problem. Given a weighted undirected graph $G = (V, E)$, a *cut* $C = (S, V \setminus S)$ where $\emptyset \subsetneq S \subsetneq V$, is a partition of vertices into two non-empty sets. The *weight* of a cut, denoted by $w(C)$, is defined to be the sum of the edge weights crossing C; i.e., $w(C) = \sum_{u \in S, v \notin S} w(uv)$. Throughout the chapter, we use λ to denote the weight of the minimum cut. A $(1+\epsilon)$-approximate minimum cut is a cut C whose weight $w(C)$ is such that $\lambda \leq w(C) \leq (1+\epsilon)\lambda$. The (approximate) minimum cut problem is to find a cut $C = (S, V \setminus S)$ with approximately the minimum weight. In the distributed setting, this means that nodes in S should output 1 while other nodes output 0.

Graph-Theoretic Notations. For $G = (V, E)$, we define $V(G) = V$ and $E(G) = E$. When we analyze the correctness of our algorithms, we will always treat G as an *unweighted multi-graph* by replacing each edge e with $w(e)$ by $w(e)$ copies of e with weight one. We note that this assumption is used only in the analysis, and in particular we still allow only $O(\log n)$ bits to be communicated through edge e in each round of the algorithm (regardless of $w(e)$). For any cut $C = (S, V \setminus S)$, let $E(C)$ denote the set of edges crossing between S and $V \setminus S$ in the multi-graph; thus $w(C) = |E(C)|$. Given an edge set $F \subseteq E$, we use G/F to denote the graph obtained by contracting every edge in F. Given a partition $\mathcal{P}$ of nodes in G, we use $G/\mathcal{P}$ to denote the graph obtained by contracting each set in $\mathcal{P}$ into one node. Note that $E(G/\mathcal{P})$ may be viewed as the set of edges in G that cross between different sets in $\mathcal{P}$. For any $U \subseteq V$, we use $G \mid U$ to denote the subgraph of G induced by nodes in U. For convenience, we use the subscript $*_H$ to denote the quantity $*$ of H; for example, λ_H denote the value of the minimum cut of the graph H. A quantity without a subscript refer to the quantity of G, the input graph.

6.3 Distributed Algorithm for Finding a Cut that 1-Respects a Tree

In this section, we solve the following problem: Given a spanning tree T on a network G rooted at some node r, we want to find an edge in T such that when we cut it, the cut defined by edges connecting the two connected component of T is smallest. To be precise, for any node v, define $v^{\downarrow}$ to be the set of nodes that are descendants of v in T, including v.

Let $C_v = (v^{\downarrow}, V \setminus v^{\downarrow})$. The problem is then to compute $c^* = \min_{v \in V(G)} w(C_v)$. The main result of this section is the following.

Theorem 6.3.1. *There is an $O(D+n^{1/2}\log^* n)$-time distributed algorithm that can compute c^* as well as find a node v such that $c^* = w(C_v)$.*

In fact, at the end of our algorithm every node v knows $w(C_v)$. Our algorithm is inspired by the following observation used in Karger's dynamic programming [97]. For any node v, let $\delta(v)$ be the weighted degree of v, i.e. $\delta(v) = \sum_{u \in V(G)} w(u, v)$. Let $\rho(v)$ denote the total weight of edges whose end-points' least common ancestor in T is v. Let $\delta^{\downarrow}(v) = \sum_{u \in v^{\downarrow}} \delta(u)$ and $\rho^{\downarrow}(v) = \sum_{u \in v^{\downarrow}} \rho(u)$.

Lemma 6.3.2 (Karger [97] (Lemma 5.9))**.** $w(C_v) = \delta^{\downarrow}(v) - 2\rho^{\downarrow}(v)$.

Our algorithm will make sure that every node v knows $\delta^{\downarrow}(v)$ and $\rho^{\downarrow}(v)$. By Theorem 6.3.2, this will be sufficient for every node v to compute $w(C_v)$. The algorithm is divided in several steps, as follows.

Step 1: Partition T into Fragments and Compute "Fragment Tree" T_F. We use the algorithm of Kutten and Peleg [114, Section 3.2] to partition nodes in tree T into $O(\sqrt{n})$ subtrees, where each subtree has $O(\sqrt{n})$ diameter[2] (every node knows which edges incident to it are in the subtree containing it). This algorithm takes $O(n^{1/2}\log^* n + D)$ time. We call these subtrees *fragments* and denote them by $F_1, \ldots, F_k$, where $k = O(\sqrt{n})$. For any i, let $\mathrm{ID}(F_i) = \min_{u \in F_i} \mathrm{ID}(u)$ be the *ID of F_i*. We can assume that every node in F_i knows $\mathrm{ID}(F_i)$. This can be achieved in $O(\sqrt{n})$ time (the running time is independent of D) by a communication within each fragment. Figure 3a illustrates the tree T (marked by black lines) with fragments (defined by triangular regions).

Let T_F be a rooted tree obtained by contracting nodes in the same fragment into one node. This naturally defines the child-parent relationship between fragments (e.g. the fragments labeled (5), (6), and (7) in Figure 3b are children of the fragment labeled (0)). Let the

[2]To be precise, we compute a $(\sqrt{n}+1, O(\sqrt{n}))$ *spanning forest*, where each tree in the spanning forest contains at least $\sqrt{n}+1$ nodes and has diameter bounded $O(\sqrt{n})$. Also note that we in fact do not need this algorithm since we obtain T by using Kutten and Peleg's MST algorithm, which already computes the $(\sqrt{n}+1, O(\sqrt{n}))$ spanning forest as a subroutine. See [114] for details.

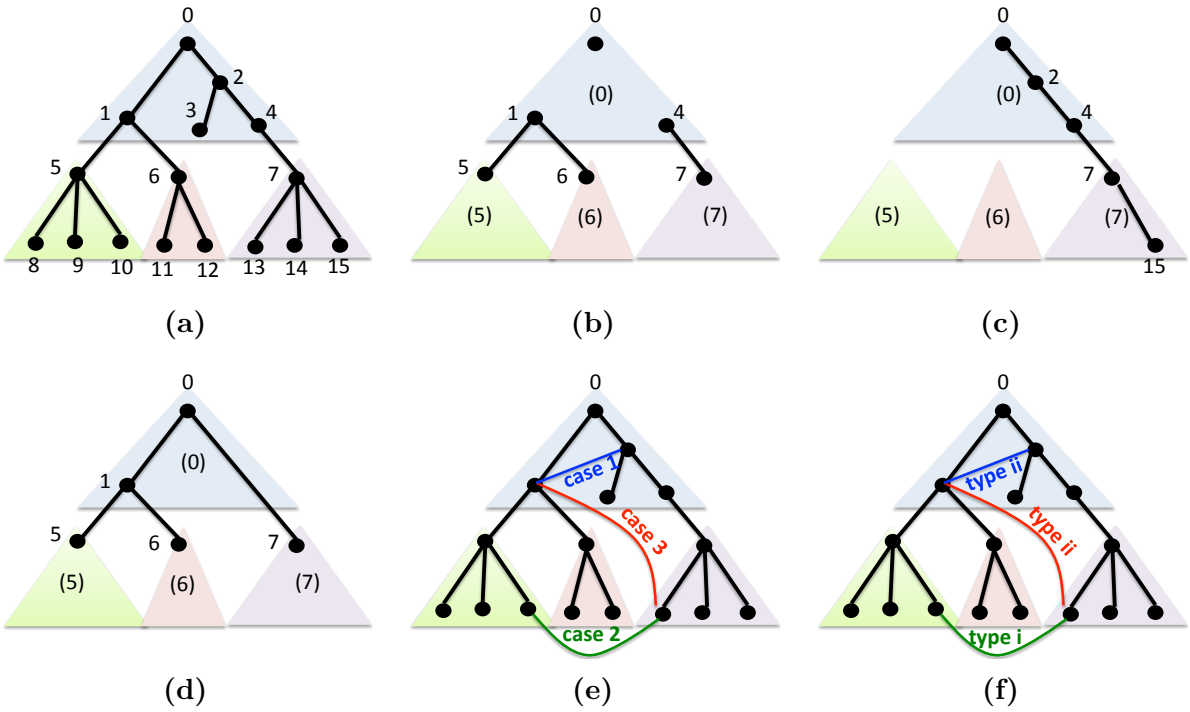

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
(a)
0
(0)
1
4
5
6
7
(5)
(6)
(7)
(b)
0
(0)
2
4
7
(5)
(6)
(7)
15
(c)
0
(0)
1
5
6
7
(5)
(6)
(7)
(d)
0
case 1
case 3
case 2
(e)
0
type ii
type ii
type i
(f)

Figure 3

root of any fragment F_i, denoted by r_i, be the node in F_i that is nearest to the root r in T. We now make every node know T_F: Every "inter-fragment" edge, i.e. every edge (u, v) such that u and v are in different fragments, either node u or v broadcasts this edge and the IDs of fragments containing u and v to the whole network. This step takes $O(\sqrt{n} + D)$ time since there are $O(\sqrt{n})$ edges in T that link between different fragments and so they can be collected by pipelining. Note that this process also makes every node know the roots of all fragments since, for every inter-fragment edge (u, v), every node knows the child-parent relationship between two fragments that contain u and v.

Step 2: Compute Fragments in Subtrees of Ancestors. For any node v let $F(v)$ be the set of fragments $F_i \subseteq v^{\downarrow}$. For any node v in any fragment F_i, let $A(v)$ be the set of ancestors of v in T that are in F_i or the *parent* fragment of F_i (also let $A(v)$ contain v). (For example, Figure 3c shows $A(15)$.) We emphasize that $A(v)$ does not contain ancestors of v in the fragments that are neither F_i nor the parent of F_i. The goal of this step is to make every node v know (i) $A(v)$ and (ii) $F(u)$ for all $u \in A(v)$.

First, we make every node v know $F(v)$: for every fragment F_i we aggregate from the leaves to the root of F_i (i.e. upcast) the list of child fragments of F_i. This takes $O(\sqrt{n} + D)$ time since there are $O(\sqrt{n})$ fragments to aggregate and each fragment has diameter $O(\sqrt{n})$. In this process every node v receives a list of child fragments of F_i that are contained in $v^{\downarrow}$. It can then use T_F to compute fragments that are descendants of these child fragments, and thus compute *all* fragments contained in $v^{\downarrow}$.

Next, we make every node v in every fragment F_i know $A(v)$: every node u sends a message containing its ID down the tree T until this message reaches the leaves of the child fragments of F_i. Since each fragment has diameter $O(\sqrt{n})$ and the total number of messages sent inside each fragment is $O(\sqrt{n})$, this process takes $O(\sqrt{n})$ time (the running time is independent of D). With the following minor modifications, we can also make every node v know $F(u)$ (the fragment that u is in) for all $u \in A(v)$: Initially every node u sends a message (u, F'), for every $F' \in F(u)$, to its children. Every node u that receives a message (u', F') from its parent sends this message further to its children *if* $F' \notin F(u)$. (A message (u', F') that a node u sends to its children should be interpreted as "u' is the lowest ancestor of u such that $F' \in F(u')$".)

Step 3: Compute $\delta^{\downarrow}(v)$. For every fragment F_i, we let $\delta(F_i) = \sum_{v \in F_i} \delta(v)$ (i.e. the sum of degree of nodes in F_i). For every node v in every fragment F_i, we will compute $\delta^{\downarrow}(v)$ by separately computing (i) $\sum_{u \in F_i \cap v^{\downarrow}} \delta(u)$ and (ii) $\sum_{F_j \in F(v)} \delta(F_j)$. The first quantity can be computed in $O(\sqrt{n})$ time (regardless of D) by computing the sum within F_i (every node v sends the sum $\sum_{u \in F_i \cap v^{\downarrow}} \delta(u)$ to its parent). To compute the second quantity, it suffices to make every node know $\delta(F_i)$ for all i since every node v already knows $F(v)$. To do this, we make every root r_i know $\delta(F_i)$ in $O(\sqrt{n})$ time by computing the sum of degree of nodes within each F_i. Then, we can make every node know $\delta(F_i)$ for all i by letting r_i broadcast $\delta(F_i)$ to the whole network.

Step 4: Compute Merging Nodes and T'_F. We say that a node v is a *merging node* if there are two distinct children x and y of v such that both $x^{\downarrow}$ and $y^{\downarrow}$ contain some fragments. In other words, it is a point where two fragments "merge". For example, nodes 0 and 1 in Figure 3a are merging nodes since the subtree rooted at node 0 (respectively node 1) contains fragments (5), (6), and (7) (respectively (5) and (6)).

Let T'_F be the following tree: Nodes in T'_F are both roots of fragments (r_i's) and merging nodes. The parent of each node v in T'_F is its lowest ancestor in T that appears in T'_F (see Figure 3d for an example). Note that every merging node has at least two children in T'_F. This shows that there are $O(\sqrt{n})$ merging nodes. The goal of this step is to let every node know T'_F.

First, note that every node v can easily know whether it is a merging node or not in one round by checking, for each child u, whether $u^{\downarrow}$ contains any fragment (i.e. whether $F(u) = \emptyset$). The merging nodes then broadcast their IDs to the whole network. (This takes $O(\sqrt{n})$ time since there are $O(\sqrt{n})$ merging nodes.) Note further that every node v in T'_F knows its parent in T'_F because its parent in T'_F is one of its ancestors in $A(v)$. So, we can make every node know T'_F in $O(\sqrt{n}+D)$ rounds by letting every node in T'_F broadcast the edge between itself and its parent in T'_F to the whole network.

Step 5: Compute $\rho^{\downarrow}(v)$. We now count, for every node v, the number of edges whose least common ancestors (LCA) of their end-nodes are v. For every edge (x, y) in G, we claim

that x and y can compute the LCA of (x, y) by exchanging $O(\sqrt{n})$ messages through edge (x, y). Let z denote the LCA of (x, y). Consider three cases (see Figure 3e).

Case 1: First, consider when x and y are in the same fragment, say F_i. In this case we know that z must be in F_i. Since x and y have the lists of their ancestors in F_i, they can find z by exchanging these lists. There are $O(\sqrt{n})$ nodes in such list so this takes $O(\sqrt{n})$ time. In the next two cases we assume that x and y are in different fragments, say F_i and F_j, respectively.

Case 2: z is *not* in F_i and F_j. In this case, z is a merging node such that $z^{\downarrow}$ contains F_i and F_j. Since both x and y know T'_F and their ancestors in T'_F, they can find z by exchanging the list of their ancestors in T'_F. There are $O(\sqrt{n})$ nodes in such list so this takes $O(\sqrt{n})$ time.

Case 3: z is in F_i (the case where z is in F_j can be handled in a similar way). In this case $z^{\downarrow}$ contains F_j. Since x knows $F(x')$ for all its ancestors x' in F_i, it can compute its lowest ancestor x'' such that $F(x'')$ contains F_j. Such ancestor is the LCA of (x, y).

Now we compute $\rho^{\downarrow}(v)$ for every node v by splitting edges (x, y) whose LCA is v into two types (see Figure 3f): (i) those that x and y are in different fragments from v, and (ii) the rest. For (i), note that v must be a merging node. In this case one of x and y creates a message $\langle v \rangle$. We then count the number of messages of the form $\langle v \rangle$ for every merging node v by computing the sum along the breadth-first search tree of G. This takes $O(\sqrt{n} + D)$ time since there are $O(\sqrt{n})$ merging nodes. For (ii), the node among x and y that is in the same fragment as v creates and keeps a message $\langle v \rangle$. Now every node v in every fragment F_i counts the number of messages of the form $\langle v \rangle$ in $v^{\downarrow} \cap F_i$ by computing the sum through the tree F_i. Note that, to do this, every node u has to send the number of messages of the form $\langle v \rangle$ to its parent, for all v that is an ancestor of u in the same fragment. There are $O(\sqrt{n})$ such ancestors, so we can compute the number of messages of the form $\langle v \rangle$ for every node v *concurrently* in $O(\sqrt{n})$ time by pipelining.

6.4 Minimum Cut Algorithms

This section is organized as follows. In Section 6.4.1, we review properties of the greedy tree packing as analyzed by Thorup [165]. We use these properties to develop a $(1 + \epsilon)$-

approximation algorithm in Section 6.4.2. We show how to efficiently implement this algorithm in the distributed setting in Section 6.4.3 and in the sequential setting in Section 6.4.4.

6.4.1 A Review of Thorup's Work on Tree Packings

In this section, we review the duality connection between the tree packing and the partition of a graph as well as their properties from Thorup's work [165].

A *tree packing* $\mathcal{T}$ is a multiset of spanning trees. The *load* of an edge e with respect to $\mathcal{T}$, denoted by $\mathcal{L}^{\mathcal{T}}(e)$, is the number of trees in $\mathcal{T}$ containing e. Define the *relative load* to be $\ell^{T}(e) = \mathcal{L}^{\mathcal{T}}(e)/|\mathcal{T}|$. A tree packing $\mathcal{T} = \{T_1, \ldots, T_k\}$ is *greedy* if each T_i is a minimum spanning tree with respect to the loads induced by $\{T_1, \ldots, T_{i-1}\}$.

Given a tree packing $\mathcal{T}$, define its *packing value* $\texttt{pack\_val}(\mathcal{T}) = 1/\max_{e\in E} \ell^{\mathcal{T}}(e)$. The packing value can be viewed as the total weight of a fractional tree packing, where each tree has weight $1/\max_{e\in E} \mathcal{L}^{\mathcal{T}}(e)$. Thus, the sum of the weight over the trees is $|T|/\max_{e\in E} \mathcal{L}^{\mathcal{T}}(e)$, which is $\texttt{pack\_val}(\mathcal{T})$. Given a partition $\mathcal{P}$, define its *partition value* $\texttt{part\_val}(\mathcal{P}) = \frac{|E(G/\mathcal{P})|}{|\mathcal{P}|-1}$. For any tree packing $\mathcal{T}$ and partition $\mathcal{P}$, we have the weak duality:

$$
\begin{aligned}
\texttt{pack\_val}(\mathcal{T}) &= \frac{1}{\max_{e\in E} \ell^{\mathcal{T}}(e)} \\
&\leq \frac{1}{\max_{e\in E(G/\mathcal{P})} \ell^{\mathcal{T}}(e)} \\
&\leq \frac{|E(G/\mathcal{P})|}{\sum_{e\in E(G/\mathcal{P})} \ell^{\mathcal{T}}(e)} && \text{(since max} \geq \text{avg)} \\
&\leq \frac{|E(G/\mathcal{P})|}{|\mathcal{P}|-1} \\
&\quad \text{(since each } T \in \mathcal{T} \text{ contains at least } |\mathcal{P}|-1 \text{ edges crossing } \mathcal{P}) \\
&= \texttt{part\_val}(\mathcal{P})
\end{aligned}
$$

The Nash-Williams-Tutte Theorem [135, 168] states that a graph G contains $\min_{\mathcal{P}} \lfloor \frac{|E(G/\mathcal{P})|}{|\mathcal{P}|-1} \rfloor$ edge-disjoint spanning trees. Construct the graph G' by making $|\mathcal{P}|-1$ parallel edges for every edge in G. It follows from the Nash-Williams-Tutte Theorem that G' has exactly

$|E(G/\mathcal{P})|$ edge-disjoint spanning trees. By assigning each spanning tree a weight of $1/(|\mathcal{P}|-1)$, we get a tree packing in G whose packing value equals to $\frac{|E(G/\mathcal{P})|}{|\mathcal{P}|-1}$. Therefore,

$$\max_{\mathcal{T}} \texttt{pack\_val}(\mathcal{T}) = \min_{\mathcal{P}} \texttt{part\_val}(\mathcal{P}).$$

We will denote this value by Φ. Let $\mathcal{T}^*$ and $\mathcal{P}^*$ denote a tree packing and a partition with $\texttt{pack\_val}(\mathcal{T}^*) = \Phi$ and $\texttt{part\_val}(\mathcal{P}^*) = \Phi$. Karger [97] showed the following relationship between Φ and λ (recall that λ is the value of the minimum cut).

Lemma 6.4.1. $\lambda/2 < \Phi \leq \lambda$

Proof. $\Phi \leq \lambda$ is obvious because a minimum cut is a partition with partition value exactly λ. Consider an optimal partition $\mathcal{P}^*$. Let $C_{\min}$ be the smallest cut induced by the components in $\mathcal{P}^*$. We have

$$\lambda \leq w(C_{\min}) \leq \frac{\sum_{S\in\mathcal{P}^*} |E(S, V\setminus S)|}{|\mathcal{P}^*|} \leq \frac{2|E(G/\mathcal{P}^*)|}{|\mathcal{P}^*|} < 2\Phi. \qquad \square$$

Thorup [165] defined the *ideal relative loads* $\ell^*(e)$ on the edges of G by the following.

1. Let $\mathcal{P}^*$ be an optimal partition with $\texttt{part\_val}(\mathcal{P}^*) = \Phi$.
2. For all $e \in G/\mathcal{P}^*$, let $\ell^*(e) = 1/\Phi$.
3. For each $S \in \mathcal{P}^*$, recurse the procedure on the subgraph $G|S$.

Define the following notations:

$$E^X_{\circ\delta} = \{e \in E \mid \ell^X(e) \ \circ \ \delta\}$$

where X can be $\mathcal{T}$ or $*$, and $\circ$ can be $<$, $>$, $\leq$, $\geq$, or $=$. For example, $E^*_{<\delta}$ denote the set of edges with ideal relative loads smaller than δ.

Lemma 6.4.2 ([165], Lemma 14)**.** *The values of* Φ *are non-decreasing in the sense that for each* $S \in P^*, \Phi_{G|S} \geq \Phi$

Corollary 6.4.3. *Let $0 \leq l \leq 1/\Phi$. Each component H of the graph $(V, E^*_{\leq l})$ must have edge-connectivity of at least Φ.*

Proof. Accroding to how the ideal relative load was defined and Lemma 6.4.2, we must have $\Phi_H \geq \Phi$. By Lemma 6.4.1, $\lambda_H \geq \Phi_H \geq \Phi$. □

Thorup showed that the relative loads of a greedy tree packing with a sufficient number of trees approximate the ideal relative loads, due to the fact that greedily packing the trees simulates the multiplicative weight update method. He showed the following lemma.

Lemma 6.4.4 ([165], Proposition 16)**.** *A greedy tree packing $\mathcal{T}$ with at least $(6\lambda \ln m)/\epsilon^2$ trees, $\epsilon < 2$ has $|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \epsilon/\lambda$ for all $e \in E$.*

6.4.2 Algorithms

In this section, we show how to approximate the value of the minimum cut as well as how to find an approximate minimum cut.

Algorithm for computing minimum cut value. The main idea is that if we have a nearly optimal tree packing, then either λ is close to 2Φ or all the minimum cuts are crossed exactly once by some trees in the tree packing.

Lemma 6.4.5. *Suppose that $\mathcal{T}$ is a greedy tree packing with at least $6\lambda \ln m/\epsilon^2$ trees, then $\lambda \leq (2+\epsilon) \cdot \texttt{pack\_val}(\mathcal{T})$. Furthermore, if there is a minimum cut C such that it is crossed at least twice by every tree in $\mathcal{T}$, then $(2+\epsilon) \cdot \texttt{pack\_val}(\mathcal{T}) \leq (1+\epsilon/2)\lambda$.*

Proof. By Lemma 6.4.4 and Lemma 6.4.1, $1/\texttt{pack\_val}(\mathcal{T}) \leq 1/\texttt{pack\_val}(\mathcal{T}^*) + \epsilon/\lambda \leq 2/\lambda + \epsilon/\lambda$. Therefore, $\lambda \leq (2+\epsilon) \cdot \texttt{pack\_val}(\mathcal{T})$.

If each tree in $\mathcal{T}$ crosses C at least twice, we have $\sum_{e \in C} \ell^{\mathcal{T}}(e) \geq 2$. Therefore,

$$\begin{aligned} 2/\lambda \leq \sum_{e \in C} \ell^{\mathcal{T}}(e)/w(C) \leq \max_{e \in C} \ell^{\mathcal{T}}(e) \leq 1/\texttt{pack\_val}(\mathcal{T}) \\ 2 \cdot \texttt{pack\_val}(\mathcal{T}) \leq \lambda \end{aligned} \tag{6.1}$$

multiplying both sides by $(1+\epsilon/2)$, we have $(2+\epsilon) \cdot \texttt{pack\_val}(\mathcal{T}) \leq (1+\epsilon/2)\lambda$. □

Using Lemma 6.4.5, we can obtain a simple algorithm for $(1+\epsilon)$-approximating the minimum cut *value*. First, greedily pack $\Theta(\lambda \log n/\epsilon^2)$ trees and compute the minimum cut that 1-respects the trees (using our algorithm in Section 6.3). Then, output the smaller value between the minimum cut found and $(2+\epsilon) \cdot \texttt{pack\_val}(\mathcal{T})$. The running time is discussed in Section 6.4.3.

Algorithm for finding a minimum cut. More work is needed to be done if we want to *find* the $(1+\epsilon)$-approximate minimum cut (i.e. each node wants to know which side of the cut it is on). Let $\epsilon' = \Theta(\epsilon)$ be such that $(1-2\epsilon') \cdot (1-\epsilon') = 1/(1+\epsilon)$. Let $l_a = (1-2\epsilon')/\texttt{pack\_val}(\mathcal{T})$. We describe our algorithm in Algorithm 12.

1: Find a greedy tree packing $\mathcal{T}$ with $(6\lambda \ln m)/\epsilon'^2$ trees in G.
2: Let C^* be the minimum cut among cuts that 1-respect a tree in $\mathcal{T}$.
3: Let $l_a = (1-2\epsilon')/\texttt{pack\_val}(\mathcal{T})$.
4: **if** $(V, E^{\mathcal{T}}_{<l_a})$ has more than $(1-\epsilon')|V|$ components **then**
5: Let $C_{\min}$ be the smallest cut induced by the components in $(V, E^{\mathcal{T}}_{<l_a})$.
6: **else**
7: Let $C_{\min}$ be the cut returned by Approx-Min-Cut$(G/E^{T}_{<l_a})$.
8: **end if**
9: Return the smaller cut between C^* and $C_{\min}$.

Algorithm 12: Approx-Min-Cut(G)

The main result of this subsection is the following theorem.

Theorem 6.4.6. *Algorithm 12 gives a $(1+\epsilon)$-approximate minimum cut.*

The rest of this subsection is devoted to proving Theorem 6.4.6. First, observe that if a minimum cut is crossed exactly once by a tree in $\mathcal{T}$, then C^* must be a minimum cut. Otherwise, C is crossed at least twice by every tree in $\mathcal{T}$. In this case, we will show that the edges of every minimum cut will be included in $E^{\mathcal{T}}_{\geq l_a}$. As a result, we can contract each connected component in the partition $(V, E^{\mathcal{T}}_{<l_a})$ without contracting any edges of the minimum cuts.

If $(V, E^{\mathcal{T}}_{<l_a})$ has at most $(1-\epsilon')|V|$ components, then we contract each component and then recurse. The recursion can only happen at most $O(\log n/\epsilon)$ times, since the number of nodes reduces by a $(1-\epsilon')$ factor in each level. On the other hand, if $(V, E^{\mathcal{T}}_{<l_a})$ has more than $(1-\epsilon')|V|$ components, then we will show that one of the components induces an approximate minimum cut.

Lemma 6.4.7. *Let C be a minimum cut such that C is crossed at least twice by every tree in $\mathcal{T}$. For all $e \in C$, $\ell^{\mathcal{T}}(e) \geq (1-2\epsilon')/\texttt{pack\_val}(\mathcal{T})$.*

Proof. The idea is to show that if an edge in $E(C)$ has a small relative load, then the average relative load over the edges in $E(C)$ will also be small. However, since each tree cross $E(C)$ twice, the average relative load should not be too small. Otherwise, a contradiction will occur.

Let $l_0 = \min_{e\in C} \ell^*(e)$ be the minimum ideal relative load over the edges in $E(C)$. Consider the subgraph $(V, E^*_{\leq l_0})$. $E(C)$ must contain some edges in a component of $(V, E^*_{\leq l_0})$, say component H. Notice that two endpoints of an edge in a minimum cut must lie on different sides of the cut. Therefore, $C \cap H$ must be a cut of H. By Corollary 6.4.3, $w(C \cap H) \geq \Phi$. Therefore, more than Φ edges in C have ideal relative loads equal to l_0. Since the maximum relative load of an edge is at most $\frac{1}{\Phi}$, $\sum_{e\in C} \ell^{\mathcal{T}^*}(e) \leq \Phi\cdot l_0 + (\lambda - \Phi)\cdot\frac{1}{\Phi} = \Phi\cdot l_0 + \frac{\lambda}{\Phi} - 1 < \Phi\cdot l_0 + 1$, where the last inequality follows by Lemma 6.4.1 that $\lambda < 2\Phi$.

On the other hand, since each tree in $\mathcal{T}$ crosses C at least twice, $\sum_{e\in C} \ell^{\mathcal{T}}(e) \geq 2$. By Lemma 6.4.4, $\sum_{e\in C} \ell^*(e) \geq 2-\epsilon'$. Therefore, $\Phi \cdot l_0 + 1 > 2 - \epsilon'$, which implies

$$\begin{aligned} l_0 &\geq (1-\epsilon')\cdot\frac{1}{\Phi} > \frac{1}{\Phi} - \frac{2\epsilon'}{\lambda} && \lambda < 2\Phi \\ &\geq 1/\texttt{pack\_val}(\mathcal{T}) - \frac{3\epsilon'}{\lambda} && \text{By Lemma 6.4.4} \end{aligned}$$

Therefore, by Lemma 6.4.4 again, for any $e \in E(C)$, $\ell^{\mathcal{T}}(e) \geq l_0 - \epsilon'/\lambda > 1/\texttt{pack\_val}(\mathcal{T}) - 4\epsilon'/\lambda \geq (1-2\epsilon')/\texttt{pack\_val}(\mathcal{T})$, where the last inequality follows from equation (6.1). □

Lemma 6.4.8. *Let $C_{\min}$ be the smallest cut induced by the components in $(V, E^{\mathcal{T}}_{<l_a})$. If $(V, E^{\mathcal{T}}_{<l_a})$ contains at least $(1-\epsilon')|V|$ components, then $w(C_{\min}) \leq (1+\epsilon)\lambda$.*

Proof. Let $comp(V, E^{\mathcal{T}}_{<l_a})$ denote the collection of connected components in $(V, E^{\mathcal{T}}_{<l_a})$, and n', the number of connected components in $(V, E^{\mathcal{T}}_{<l_a})$. By an averaging argument, we have

$$w(C_{\min}) \leq \frac{\sum_{S \in comp(V, E^{\mathcal{T}}_{<l_a})} |E(S, V \setminus S)|}{n'} = \frac{2|E(G/E^{\mathcal{T}}_{<l_a})|}{n'} \leq \frac{2|E(G/E^{\mathcal{T}}_{<l_a})|}{(1-\epsilon') \cdot |V|} \tag{6.2}$$

Next we will bound $|E(G/E^{\mathcal{T}}_{<l_a})|$. Note that for each $e \in E(G/E^{\mathcal{T}}_{<l_a})$, $\ell^{\mathcal{T}}(e) \geq (1 - 2\epsilon')/\texttt{pack\_val}(\mathcal{T})$.

$$\begin{aligned} \sum_{e \in E(G/E^{\mathcal{T}}_{<l_a})} \ell^{\mathcal{T}}(e) &\geq |E(G/E^{\mathcal{T}}_{<l_a})| \cdot (1-2\epsilon') \cdot \left(\frac{1}{\texttt{pack\_val}(\mathcal{T})}\right) \\ &\geq |E(G/E^{\mathcal{T}}_{<l_a})| \cdot (1-2\epsilon') \cdot \frac{2}{\lambda}\,. \qquad \text{(by (6.1))} \end{aligned} \tag{6.3}$$

On the other hand,

$$\sum_{e \in E(G/E^{\mathcal{T}}_{<l_a})} \ell^{T}(e) \leq |V| - 1, \tag{6.4}$$

since each tree in $\mathcal{T}$ contains $|V| - 1$ edges. Eqaution (6.3) and (6.4) together imply that

$$|E(G/E^{\mathcal{T}}_{<l_a})| \leq \frac{\lambda \cdot |V|}{2(1-2\epsilon')}.$$

By plugging this into (6.2), we get that

$$w(C_{\min}) \leq \frac{\lambda}{(1-2\epsilon')(1-\epsilon')} \leq (1+\epsilon)\lambda\,. \qquad \square$$

6.4.3 Distributed Implementation

In this section, we describe how to implement Algorithm 12 in the distributed setting. To compute the tree packing $\mathcal{T}$, it is straightforward to apply $|\mathcal{T}|$ minimum spanning tree computations with edge weights equal to their current loads. This can be done in $O(|\mathcal{T}|(D + \sqrt{n}\log^* n))$ rounds by using the algorithm of Kutten and Peleg [114].

We already described how to computes the minimum cut that 1-respects a tree in $O(D +$

$\sqrt{n}\log^* n)$ rounds in Section 6.3. To compute l_a, it suffices to compute $\texttt{pack\_val}(\mathcal{T})$. To do this, each node first computes the largest relative load among the edges incident to it. By using the upcast and downcast techniques, the maximum relative load over all edges can be aggregated and broadcast to every node in $O(D)$ time. Therefore, we can assume that every node knows l_a now. Now we have to determine whether $(V, E^{\mathcal{T}}_{<l_a})$ has more than $(1-\epsilon')|V|$ components or not. This can be done by first removing the edges incident to each node with relative load at least l_a. Then label each node with the smallest ID of its reachable nodes by using Thurimella's connected component identification algorithm [166] in $O(D+\sqrt{n}\log^* n)$ rounds. The number of nodes whose label equals to its ID is exactly the number of connected component of the subgraph. This number can be aggregated along the BFS tree in $O(D)$ rounds after every node is labeled.

If $(V, E^{\mathcal{T}}_{<l_a})$ has more than $(1-\epsilon')|V|$ components, then we will compute the cut values induced by each component of $(V, E^{\mathcal{T}}_{<l_a})$. We show that it can be done in $O(D+\sqrt{n})$ rounds in Section 6.5. On the contrary, if $(V, E^{\mathcal{T}}_{<l_a})$ has less than $(1-\epsilon')|V|$ components, then we will contract the edges with load less than l_a and then recurse. The contraction can be easily implemented by setting the weights of the edges inside contracted components to be -1, which is strictly less than the load of any edges. The MST computation will automatically treat them as contracted edges, since an MST must contain exactly $n'-1$ edges with weights larger than -1, where n' is the number of connected components. [3]

Time analysis. Suppose that we have packed t spanning trees throughout the entire algorithm, the running time will be $O(t(D+\sqrt{n}\log^* n))$. Note that $t = O(\epsilon^{-3}\lambda\log^2 n)$, because we pack at most $O(\epsilon^{-2}\lambda\log n)$ spanning trees in each level of the recursion and there can be at most $O(\epsilon^{-1}\log n)$ levels, since the number of nodes reduces by a $(1-\epsilon')$ factor in each level. The total running time is $O(\epsilon^{-3}\lambda\log^2 n\cdot(D+\sqrt{n}\log^* n))$.

Dealing with graphs with high edge connectivity. For graphs with $\lambda = \omega(\epsilon^{-2}\log n)$, we can use the well-known sampling result from Karger [96] to construct a subgraph H that perserves the values of all the cuts within a $(1\pm\epsilon)$ factor (up to a scaling) and has $\lambda_H = O(\epsilon^{-2}\log n)$. Then we run our algorithm on H.

[3] We note that the MST algorithm of [114] allows negative-weight edges.

Lemma 6.4.9 ([95], Corollary 2.4)**.** *Let G be any graph with minimum cut λ and let $p = 2(d+2)(\ln n)/(\epsilon^2\lambda)$. Let $G(p)$ be a subgraph of G with the same vertex set, obtained by including each edge of G with probability p independently. Then the probability that the value of some cut in $G(p)$ has value more than $(1+\epsilon)$ or less than $(1-\epsilon)$ times its expected value is $O(1/n^d)$.*

In particular, let $\epsilon' = \Theta(\epsilon)$ such that $(1+\epsilon) = (1+\epsilon')^2/(1-\epsilon')$. First we will compute λ', a 3-approximation of λ, by using Ghaffari and Kuhn's algorithm. Let $p = 6(d+2)\ln n/(\epsilon'^2\lambda')$ and $H = G(p)$. Since p is at least $2(d+2)\ln n/(\epsilon'^2\lambda)$, by Lemma 6.4.9, for any cut C, w.h.p. $(1-\epsilon')p \cdot w_G(C) \leq w_{H_i}(C) \leq (1+\epsilon')p \cdot w_G(C)$. Let C^* be the $(1+\epsilon')$-approximate minimum cut we found in H. We have that w.h.p. for any other cut C',

$$w_G(C^*) \leq \frac{1}{p}\cdot\frac{w_{H_i}(C^*)}{1-\epsilon'} \leq \frac{1}{p}\cdot\frac{(1+\epsilon')\lambda_H}{1-\epsilon'} \leq \frac{1}{p}\cdot\frac{(1+\epsilon')w_{H_i}(C')}{1-\epsilon'} \leq \frac{(1+\epsilon')^2}{1-\epsilon'}\cdot w_G(C') = (1+\epsilon)w_G(C')$$

Thus, we will find an $(1+\epsilon)$-approximate minimum cut in $O(\epsilon^{-5}\log^3 n(D+\sqrt{n}\log^* n))$ rounds.

Computing the exact minimum cut. To find the exact minimum cut, first we will compute a 3-approximation of λ, λ', by using Ghaffari and Kuhn's algorithm [65] in $O(\lambda \log n \log\log n(D+\sqrt{n}\log^* n))$ rounds.[4] Now since $\lambda \leq \lambda' \leq 3\lambda$, by applying our algorithm with $\epsilon = 1/(\lambda'+1)$, we can compute the exact minimum cut in $O(\lambda^4 \log^2 n(D+\sqrt{n}\log^* n))$ rounds.

Estimating the value of λ. As described in Section 6.4.2, we can avoid the recursion if we just want to compute an approximation of λ without actually finding the cut. This gives an algorithm that runs in $O(\epsilon^{-2}\lambda \log n \cdot (D+\sqrt{n}\log^* n))$ time. Also, the exact value of λ can be computed in $O((\lambda^3 + \lambda\log\log n)\log n(D+\sqrt{n}\log^* n))$ rounds. Notice that the $\lambda \log\log n$ factor comes from Ghaffari and Kuhn's algorithm for approximating λ within a constant factor. Similarly, using Karger's sampling result, we can $(1+\epsilon)$-approximate the value of λ in $O(\epsilon^{-5}\log^2 n \log\log n(D+\sqrt{n}\log^* n))$ rounds.

[4]Ghaffari and Kuhn's result runs in $O(\log^2 n \log\log n(D+\sqrt{n}\log^* n))$ rounds. However, without using Karger's random sampling beforehand, it runs in $O(\lambda \log n \log\log n(D+\sqrt{n}\log^* n))$ rounds, which will be absorbed by the running time of our algorithm for the exact minimum cut.

6.4.4 Sequential Implementation

We show that Algorithm 12 can be implemented in the sequential setting in $O(\epsilon^{-3}\lambda(m + n\log n)\log n)$ time. To get the stated bound, we will show that the number of edges decreases geometrically each time we contract the graph.

Lemma 6.4.10. *If $(V, E^{\mathcal{T}}_{<l_a})$ has less than $(1-\epsilon')|V|$ components, then $|E(G/E^{\mathcal{T}}_{<l_a})| \leq |E(G)|/(1+\epsilon')$.*

Proof. Consider a component S of $(V, E^{\mathcal{T}}_{<l_a})$. Since $E(S) \subseteq E^{\mathcal{T}}_{<l_a}$ and $|T \cap E(S)| \geq |S| - 1$, we have $|S| - 1 \leq \sum_{e\in S} \ell^{\mathcal{T}}(e) < l_a|E(S)|$. By summing this inequality over all components of $(V, E^{\mathcal{T}}_{<l_a})$, we have

$$l_a|E^{\mathcal{T}}_{<l_a}| \geq |V| - |V(G/E^{\mathcal{T}}_{<l_a})| > |V| - (1-\epsilon')|V| = \epsilon'|V| \tag{6.5}$$

If we sum up the relative load over each $e \in E(G/E^{\mathcal{T}}_{<l_a})$, we have

$$l_a|E(G/E^{\mathcal{T}}_{<l_a})| \leq \sum_{e\in E(G/E^{\mathcal{T}}_{<l_a})} \ell^{\mathcal{T}}(e) \leq |V| \tag{6.6}$$

Dividing (6.5) by (6.6), we have $|E^{\mathcal{T}}_{<l_a}|/|E(G/E^{\mathcal{T}}_{<l_a})| > \epsilon'$ and therefore, $|E(G/E^{\mathcal{T}}_{<l_a})| < (|E^{\mathcal{T}}_{<l_a}| + |E(G/E^{\mathcal{T}}_{<l_a})|)/(1+\epsilon') = |E(G)|/(1+\epsilon')$. □

Let $\texttt{MST}(n, m)$ denote the time needed to find an MST in a graph with n-vertices and m-edges. Note that Karger [97] showed that the values of the cuts that 1-respect a tree can be computed in linear time. The total running time of Algorithm 12 will be

$$O\left(\epsilon'^{-2}\lambda\log n \cdot \sum_{i=0}^{\infty} \texttt{MST}(n(1-\epsilon')^i, m/(1+\epsilon')^i)\right).$$

We know that $\texttt{MST}(n, m) = O(m)$ by using the randomized linear time algorithm from [98] and notice that $\epsilon = \Theta(\epsilon')$, the running time will be at most $O(\epsilon^{-3}\lambda m\log n)$.

If the graph is dense or the cut value is large, we may want to use the sparsification results to reduce m or λ. First estimate λ up to a factor of 3 by using Matula's algorithm [121] that

runs in linear time. By using Nagamochi and Ibaraki's sparse certificate algorithm [132], we can get the number of edges down to $O(n\lambda)$. By using Karger's sampling result, we can bring λ down to $O(\log n/\epsilon^2)$. The total running time is therefore $O(m + \epsilon^{-7} n \log^3 n)$ (by plugging $\lambda = \log n/\epsilon^2$ and $m = n \log n/\epsilon^2$ in the running time in the previous paragraph). [5]

6.5 Finding cuts with respect to connected components

In this section, we solve the following problem. We are given a set of connected components $\{H_1, H_2, \ldots, H_k\}$ of the network G (each node knows which of its neighbors are in the same connected component), and we want to compute, for each i, the value $w(C_i)$ where C_i is the cut with respect to H_i; i.e., $C_i = (V(H_i), V(G) \setminus V(H_i))$. Every node in C_i should know $w(C_i)$ in the end. We show that this can be done in $O(n^{1/2} + D)$ rounds. The main idea is to deal with "big" and "small" components separately, where a component is big if it contains at least $n^{1/2}$ nodes and it is small otherwise. There are at most $n^{1/2}$ big components, and thus the cut value information for these components can be aggregated quickly through the BFS tree of the network. The cut value of each small component will be computed locally within the component. The detail is as follows.

First, we determine for each component H_i whether it is big or small, which can be done by simply counting the number of nodes in each component, such as the following. Initially, every node sends its ID to its neighbors in the same component. Then, for $n^{1/2} + 1$ rounds, every node sends the smallest ID it has received so far to its neighbors in the same component. For each node v, let s_v be the smallest ID that v has received after $n^{1/2} + 1$ rounds. If s_v is v's own ID, it construct a BFS tree T_v of depth at most $n^{1/2} + 1$, and use T_v to count the number of nodes in T_v. (There will be no congestion caused by this algorithm since no other node within distance $n^{1/2} + 1$ from v will trigger another BFS tree construction.) If the number of nodes in T_v is at most $n^{1/2}$, then v broadcasts to the whole network that the component containing it is small.

[5] In this case, we can also use Prim's deterministic MST algorithm without increasing the total running time. This is because Prim's algorithm runs in $O(m + n \log n)$ time, the $n \log n$ term will be absorbed by m, as we have used $m = n \log n/\epsilon^2$.

Now, to compute $w(C_i)$ for a small component H_i, we simply construct a BFS tree rooted at the node with smallest ID in C_i and compute the sum $\sum_{u \in V(H_i), v \notin V(H_i)} w(u, v)$ through this tree. To compute $w(C_i)$ for a big component H_j, we compute the sum $\sum_{u \in V(H_i), v \notin V(H_i)} w(u, v)$ thorough the BFS tree of network G. Since there are at most $n^{1/2}$ big components, this takes $O(n^{1/2} + D)$ time.

Chapter 7

Distributed Task Allocation in Ant Colonies with Individual Variation

7.1 Introduction

Social insect biology and distributed computing both study how goals are achieved without a centralized coordinator. While that in the latter, the nodes can usually perform complex computation, it is not necessarily true in the former. In this chapter, we study simple task allocation algorithms in the decentralized setting. We aim to capture the behavior of task allocation in ant colonies. We also hope it motivates the development of simple algorithms in distributed computing which are favorable for implementation.

In ant colonies, the task allocation problem involves assigning a task to each ant in a distributed way with the main goal of satisfying the demands of all tasks. The first attempt at modeling the task allocation problem from a distributed computing perspective was in [26]. In their model, each task has a static demand and the goal is for each ant to choose a task in such a way that the sum of the work units provided to each task meets its demand. Furthermore, each ant receives a binary feedback from the environment, indicating, for each task, whether the task is over-satisfied or under-satisfied. Under the assumption that the work provided by each ant is uniform, [26] shows that the ants can solve the task allocation problem in $O(|T| \log |A|)$ rounds using a constant number of states and $O(|T|)$ bits of memory per ant, where A is the set of ants and T is the set of tasks.

Biologists have also worked on formally modeling the ant task allocation process from a

distributed perspective. For example, [15] models the ants' response to the environment as a fixed threshold that determines when an ant starts and stops working on a task; ants from different castes are assumed to have different thresholds for different tasks. Also, [72] and [139] explicitly model the interaction patterns among ants and also between ants and the environment in order to define the behavior of an individual ant. Furthermore, [146] investigates the trade-off between the sensitivity to the environment and the response time of ants, and how these aspects of the task allocation process are influenced by the colony size and the rules that govern the individual ants.

In this chapter, we present a distributed task allocation algorithm that captures the individual variation of ants in terms of the work units they provide to different tasks. As discussed in [26], this problem is NP-hard; however, we provide a very simple mechanism for the ants to *approximately* satisfy the demands of each task, provided that the original set of demands is satisfiable. In particular, we show that after $O(|A|^{1-\epsilon})$ rounds, the ants converge to a solution that satisfies the demands with an $O(W|A|^{1/2+\epsilon})$ additive error with probability at least $1 - O(1/|A|^{\epsilon})$, where W is the ratio between the largest and the smallest number of work units possibly provided by the ants. The task allocation process is similar to simulated annealing, where in each round, each ant switches to the current most promising task with some probability, and that probability diminishes in each subsequent round. The current most promising task for a given ant is the task with the largest deficit (difference between the demand and the work provided already) weighted by the work units the ant is capable of providing for the task. In the case of no individual variation between the ants, this task allocation process converges to all tasks being satisfied with $\epsilon d_{\max} + o(1)$ additive error in $O(\epsilon^{-2}|T| \log |T|)$ rounds, where $d_{\max}$ is the largest demand of the tasks. Such a bound is consistent with the proposed model and conclusions of [146], which conjecture that the response time of ants in the task allocation process does not depend on the colony size or it is very insensitive to it. The main technique in our analyses is derived from the multiplicative weight update method for solving linear programs [5,150,173] with some modifications in order to apply the method to the setting of ants with limited computation and communication abilities.

7.2 Definitions and Problem Statement

Each ant $a \in A$, where $|A|$ is the set of all ants, has a state $q \in Q = \{q_\perp, q_1, q_2, \cdots, q_k\}$, where $q_\perp$ indicates that ant a is not working on any task and each state q_i, for $i \in \{1, \cdots, k\}$, indicates that ant a is working on task i. Each task $i \in T = \{1, \ldots, k\}$ has an integer energy demand d_i that represents the minimum number of ants required to work on task i in order to satisfy the task. Clearly, in order for all demands to be met, there should be sufficiently many ants in the colony.

We assume the execution of any algorithm solving the task allocation problem proceeds in synchronous rounds, starting with round 1; we use round 0 to refer to the state of the system initially, at the beginning of the execution. For each $i \in \{1, \cdots, k\} \cup \{\perp\}$ and each round r, let $A_i^{(r)}$ denote the set of ants in state q_i at the end of round r. Let $w_i^{(r)} = \sum_{a \in A_i^{(r)}} w_{ai}$ be the total energy supplied to task i, where w_{ai} is the energy provided by ant a to task i. We define $\mathbf{w}^{(r)} = (w_\perp^{(r)}, w_1^{(r)}, \ldots, w_k^{(r)})$ and $\mathbf{d} = (0, d_1, \ldots, d_k)$.

Let $X_r = (r, \mathbf{w}^{(r)}, \mathbf{d})$ denote the environment of round r. At the end of each round r, each ant a receives feedback $f(X_r, a)$ from the environment. Since randomness usually plays a role in the environment, the value of f can also be a random variable. We assume f consists of two components (f_1, f_2), where $f_1(X_r, a)$ is a local feedback from the task ant a is working on, and $f_2(X_r, a)$ is a global feedback from the environment, providing alternative tasks to which ant a may switch.

Each ant is modeled as a finite state machine with transition function $\delta : Q \times (\{0, 1\} \times T) \to Q$; in other words, each ant's new state is determined by its old state and the environment input function $f = (f_1, f_2)$. Let q be the current state of some ant a, and let q' be the resulting state of ant a after applying δ. In each round r, q' is determined as follows: $q' = q$ if $f_1(X_r, a) = 1$, and $q' = f_2(X_r, a)$ if $f_1(X_r, a) = 0$. Informally speaking, each ant continues working on its current task if the environment input is 1 (the ant is successful at the task), or switches to task $f_2(X_r, a)$ if the environment input is 0 (the ant is not successful at the task). That is, f_1 indicates whether the ant should switch, and f_2 indicates which new task the ant should choose.

Problem Statement: For each round r, $\mathbf{w}^{(r)}$ satisfies all tasks if $d_i \leq w_i^{(r)}$ for each $i \in \{1, \dots k\}$. For each round r, $\mathbf{w}^{(r)}$ satisfies all tasks with an additive error of θ, if $w_i^{(r)} \geq d_i - \theta$ for each $i \in \{1, \dots k\}$. An algorithm solves the task allocation problem (with an additive error of θ) by the end of round r, if $\mathbf{w}^{(r')}$ satisfies all tasks (with an additive error of θ) for each round $r' \geq r$.

7.3 Task Allocation with Individual Variation

We assume ants may have different capabilities at each task. Each ant $a \in A$ is associated with a weight vector $\mathbf{w}_a = (w_{a1}, \dots, w_{a|T|})$, where w_{ai} denotes the energy provided by ant a to task i. Let $\mathbf{x}_a = (x_{a1}, \dots, x_{a|T|})$ be the indicator vector of a where $x_{ai} = 1$ if ant a is working on task i, otherwise $x_{ai} = 0$. The goal is to satisfy the demands of each task i, that is, $\sum_{a \in A} w_{ai} \cdot x_{ai} \geq d_i$. However, as discussed in [26], this problem is NP-hard even if there is a centralized coordinator. We show that under a simple feedback function $f = (f_1, f_2)$, the ants converge to an approximate feasible solution, provided that the original system is fractionally satisfiable: Given the demands of the tasks and the energy vector of each ant, there exists (non-necessarily integral) $\mathbf{x}_a$ for each a such that

1. For each a, $\sum_i x_{ai} \leq 1$ and $0 \leq x_{ai} \leq 1$.
2. For each i, $\sum_{a \in A} w_{ai} \cdot x_{ai} = d_i$.

The feedback functions f_1, f_2 are defined in the following:

$$f_1(X_r, a) = \begin{cases} 0 & \text{with prob. } \frac{1}{r+1} \\ 1 & \text{with prob. } 1 - \frac{1}{r+1} \end{cases}$$

$$f_2(X_r, a) = \begin{cases} \arg\max_j w_{aj} \cdot (d_j - w_j^{(r)}) & \text{if } \max_j w_{aj} \cdot (d_j - w_j^{(r)}) \geq 0 \\ \perp & \text{otherwise} \end{cases}$$

In contrast to the busy-success model, f_1 does not depend on the demand of the task. Instead, here f_1 tells $r/(r+1)$ fraction of the ants to continue to do the same task uniformly at random. The other $1/(r+1)$ fraction of ants will switch to the task returned by f_2.

Initially, the ants are more likely to switch tasks. The system becomes more and more stable as the rounds proceed.

The feedback function $f_2(X_r, a)$ then returns the task that needs the most work to be satisfied at the end of round r, weighted by the capability of the ant at the task, i.e. $f_2(X_r, a) = \arg\max_i w_{ai} \cdot \left(d_i - w_i^{(r)}\right)$; or f_2 returns $\perp$ if no tasks needs more work. One can imagine that there is a home nest to where the foragers return at the end of every round r. The foragers have some chances of being recruited by following the trail with the heaviest pheromone. We may assume that the amount of pheromone on the trail leading to task i is a function of $w_i^{(r)}$ and d_i, in this case, $d_i - w_i^{(r)}$. An ant may respond differently to different pheromones. It responds more sensitively to the pheromone of the task it could perform better.

Our result shows that the ants converge to an assignment with $O(W|A|^{1/2+\epsilon})$ additive error $O(|A|^{1-\epsilon})$ rounds with probability $1 - O(1/|A|^{\epsilon})$, where W is the ratio between the largest energy to the smallest energy ants could provide. (W.l.o.g., we can assume the smallest energy is 1 and the largest energy is W.) Note that in the case that $d_i = \omega(W|A|^{1/2+\epsilon})$ for a task i, the work on task i converges to $(1 - o(1))d_i$ after $O(|A|^{1-\epsilon})$ rounds.

We also show a better bound in this model when the ants' weights are homogeneous and f_1 is deterministic. The deterministic version of f_1 tells $1/(r+1)$ fraction of the ants working on each task to stop. That is, f_1 can be any function such that for each $i \in T \cup \{\perp\}$, $\sum_{a \in A_i^{(r)}} f_1(X_r, a) = \lfloor w_i^{(r)} \cdot \frac{r}{r+1} \rfloor$. When $|A| = \sum_i d_i$, the ants converges to an assignment with $\epsilon d_{\max} + o(1)$ additive error after $O(\epsilon^{-2}|T| \log |T|)$ rounds, where $d_{\max}$ is the largest demand of the tasks.

Our analysis of the task allocation process is derived from the multiplicative weight update method for solving linear programs [5, 150, 173] with a couple of modifications in order to apply the method to the setting of ants with limited computation and communication abilities. For example, in our setting, ants do not explicitly memorize the weights of the multiplicative weight update method. Instead, each ant switches to the most promising task with some probability and that step corresponds to updating the weights in the multiplicative weight update method. Such a process of updating the probabilities without explicitly remembering the weights was originally introduced in greedy tree packing algorithms for minimum cuts [97, 165]. One challenge in our approach of using the multiplicative weights update method is dealing with the errors introduced after each weight update due to the

fact that ants switch tasks probabilistically. To account for these errors, we resort to the Chebyshev-type pessimistic estimator instead of the Chernoff-type pessimistic estimator as originally used in the analysis for multiplicative weight method [173]. Also, we expect that the method for determining the most promising task would be more complicated if we were to use Chernoff-type pessimistic estimator. The bound for the deterministic and homogeneous model is derived using the Chernoff-type pessimistic estimator.

Lemma 7.3.1. *Let* $W = \max_a ||\mathbf{w}_a||_\infty$ *be the maximum energy provided by the ant over each task. Under the above setting of* f_1 *and* f_2*, for any* $\epsilon \geq 0$*, if* $t = \Theta\left(|A|^{1-\epsilon}\right)$*, with probability at least* $1 - O\left(\frac{1}{|A|^\epsilon}\right)$*,* $\mathbf{w}^{(t)}$ *satisfies the demands* $\mathbf{d}$ *with an additive error of* $\Theta(W|A|^{1/2+\epsilon})$*.*

Proof. First, notice that $w_j^{(r+1)} = \sum_{a \in A_j^{(r)}} w_{aj} \cdot [f_1(X_r, a) = 1] + \sum_{a \in A}[f_2(X_r, a) = j] \cdot [f_1(X_r, a) = 0] \cdot w_{aj}$. The expectation of $w^{(r+1)}$, $\mathrm{E}[w_j^{(r+1)}] = (w_j^{(r)} \cdot r + \sum_{a \in A}[f_2(X_r, a) = j] \cdot w_{aj})/(r+1)$. We show the following inequality holds for $0 \leq t' < t$:

$$\sum_j ((t'+1)(\mathrm{E}[w_j^{(t'+1)}] - d_j))^2 \leq \sum_j (t'(w_j^{(t')} - d_j))^2 + 4(W|A|)^2. \tag{7.1}$$

$$
\begin{aligned}
&\sum_j ((t'+1)(\mathrm{E}[w_j^{(t'+1)}] - d_j))^2 \\
&= \sum_j \left(t'(w_j^{(t')} - d_j) + (\sum_a [f_2(X_{t'}, a) = j] \cdot w_{aj} - d_j) \right)^2 \\
&\leq \sum_j \Bigg((t'(w_j^{(t')} - d_j))^2 + 2(t'(w_j^{(t')} - d_j)) \cdot (\sum_a [f_2(X_{t'}, a) = j] \cdot w_{aj} - d_j) \\
&\qquad\qquad + (\sum_a [f_2(X_{t'}, a) = j] \cdot w_{aj} - d_j)^2 \Bigg) \\
&\leq \sum_j \left((t'(w_j^{(t')} - d_j))^2 + 2(t'(w_j^{(t')} - d_j)) \cdot (\sum_a [f_2(X_{t'}, a) = j] \cdot w_{aj} - d_j) \right) + 4(W|A|)^2 \\
&\leq \sum_j \left((t'(w_j^{(t')} - d_j))^2 + 2(t'(w_j^{(t')} - d_j)) \cdot (\sum_a [f_2(X_{t'}, a) = j] \cdot (\sum_{j'} x_{aj'}) \cdot w_{aj} - d_j) \right) \\
&\qquad + 4(W|A|)^2 \qquad \text{since } \textstyle\sum_{j'} x_{aj'} \leq 1 \text{ and } f_2(X_{t'}, a) = j \text{ implies } w_j^{(t')} - d_j \leq 0 \\
&\leq \sum_j \left((t'(w_j^{(t')} - d_j))^2 + 2(t'(w_j^{(t')} - d_j)) \cdot (\sum_a x_{aj} \cdot w_{aj} - d_j) \right) + 4(W|A|)^2 \\
&\qquad\qquad \text{by defn. of } \mathbf{x}_a \text{ and since } f_2(X_{t'}, a) = \max_j w_{aj} \cdot (d_j - w_j^{(t')}) \\
&= \sum_j (t'(w_j^{(t')} - d_j))^2 + 4(W|A|)^2
\end{aligned}
$$

Given $w_j^{(r)}$ for each task j, the variance (the randomness comes from the feedback function $f_1(X_r, a)$) of $w_j^{(r+1)}$, $\mathrm{Var}(w_j^{(r+1)}) \leq W^2 \cdot (w_j^{(r)} + \sum_a [f_2(X_r, a) = j]) \cdot \frac{1}{r+1} \cdot (1 - \frac{1}{r+1})$. Therefore,

$$
\sum_j \mathrm{Var}(w_j^{(r+1)}) \leq 2 \cdot W^2 \cdot |A| \cdot \frac{1}{r+1} \cdot \left(1 - \frac{1}{r+1}\right) \tag{7.2}
$$

Define the following Chebyshev-type pessimistic estimator, where t is the total number of rounds and t' iterates from 0 to t:

$$
\Phi(t') = \frac{\sum_j (t'^2)(w_j^{(t')} - d_j)^2 + 4(t - t') \cdot W^2 \cdot |A|^2 + 2t(t - t')|A|W^2}{4t^2 W^2 |A|^{1+2\epsilon}}
$$

First, note that

$$\begin{aligned}\Phi(0) &\leq \frac{4t \cdot W^2|A|^2 + 2t^2|A|W^2}{4t^2W^2|A|^{1+2\epsilon}} \\ &\leq \frac{|A|^{1-2\epsilon}}{t} + \frac{1}{2|A|^{2\epsilon}} \\ &\leq O\left(\frac{1}{|A|^{\epsilon}}\right)\end{aligned}$$

Now we will show that the expected value of $\Phi(t')$ does not increase as t' increase.

$$\mathrm{E}[\Phi(t'+1)] = \frac{\sum_j \mathrm{E}[((t'+1)(w_i^{(t'+1)} - d_i))^2] + 4(t-t'-1) \cdot W^2|A|^2 + 2t(t-t'-1)|A|W^2}{4t^2W^2|A|^{1+2\epsilon}}$$

Consider the term $\sum_j \mathrm{E}[((t'+1)(w_j^{(t'+1)} - d_j))^2]$,

$$\begin{aligned}&\sum_j \mathrm{E}[((t'+1)(w_j^{(t'+1)} - d_j))^2] \\ &= \sum_j \mathrm{E}[((t'+1)(\mathrm{E}[w^{(t'+1)}] - d_j) + (t'+1)(w_j^{(t'+1)} - \mathrm{E}[w_j^{(t'+1)}]))^2] \\ &= \sum_j ((t'+1)(\mathrm{E}[w_j^{(t'+1)}] - d_j))^2 + \sum_j (t'+1)^2 \,\mathrm{E}[(w_j^{(t'+1)} - \mathrm{E}[w_j^{(t'+1)}])^2] \\ &= \sum_j ((t'+1)(\mathrm{E}[w_j^{(t'+1)}] - d_j))^2 + \sum_j (t'+1)^2 \cdot \mathrm{Var}(w_j^{(t'+1)}) \\ &\leq \sum_j ((t'+1)(\mathrm{E}[w_j^{(t'+1)}] - d_j))^2 + (t'+1)^2 \cdot 2W^2|A|/(t'+1) && \text{by (7.2)} \\ &\leq \sum_j (t'(w_j^{(t')} - d_j))^2 + 4W^2|A|^2 + 2(t'+1)W^2 \cdot |A| && \text{by (7.1)}\end{aligned}$$

Therefore, $\mathrm{E}[\Phi(t'+1)] \leq \Phi(t')$. Repeatedly applying the argument, we have $\mathrm{E}[\Phi(t)] \leq \mathrm{E}[\Phi(t-1)] \leq \ldots \leq \Phi(0) < O\left(\frac{1}{|A|^{\epsilon}}\right)$. By Markov's inequality, with probability at least

$1 - O\left(\frac{1}{|A|^\epsilon}\right)$:

$$1 > \Phi(t) = \sum_j \left(\frac{t \cdot (w_j^{(t)} - d_j)}{2tW|A|^{1/2+\epsilon}} \right)^2$$

This implies $w_j^{(t)} \geq d_j - (2W\sqrt{|A|}^{1+\epsilon})$ for each task j. □

7.3.1 A Better Bound for Homogeneous and Deterministic Model

In this section, we show under the following assumptions, we can obtain better bounds on the convergence time and the error. First we assume that the energy provided by each ant to each task is the same, i.e. $w_{ai} = 1$ for all $a \in A$ and $i \in T$. Second, we assume f_1 behaves determinstically, that is, f_1 is an arbitrary function such that for each $i \in T \cup \{\perp\}$, $\sum_{a \in A_i^{(r)}} f_1(X_r, a) = \lfloor w_i^{(r)} \cdot (r/(r+1)) \rfloor$. Third, we assume that $|A| = \sum_i d_i$.

Lemma 7.3.2. *Under the setting with the deterministic setting of f_1, after $t = \Theta\left(\epsilon^{-2}|T| \cdot \log|T|\right)$ rounds, $\mathbf{w}^{(t)}$ satisfies the demands $\mathbf{d}$ with an additive error of $(\epsilon \cdot d_{\max} + t/|A|)$.*

Proof. For convenience, we define $x_j^{(r)} = w_j^{(r)}/|A|$ to be the fraction of ants working at task j at the end of round r. Define $\mu_j = d_j/|A|$ and $\mu = d_{\max}/|A|$. Define the following Chernoff-type pessimistic estimator:

$$\Phi(t') = \frac{e^{-\epsilon\mu(t-t')}}{(1-\epsilon)^{-(t/|A|)(t')+\mu(t-t'+1)-\epsilon\mu t}} \sum_j \frac{(1-\epsilon)^{x_j^{(t')} \cdot t'}}{(1-\epsilon)^{\mu_j(t'-1)}}$$

First, note that

$$\begin{aligned}
\Phi(0) &\leq \frac{e^{-\epsilon\mu t}}{(1-\epsilon)^{(1-\epsilon)\mu t+\mu}} \sum_j \frac{1}{(1-\epsilon)^{-\mu_j}} \\
&\leq \exp(-\epsilon\mu t - ((1-\epsilon)\mu t + \mu) \cdot \ln(1-\epsilon)) \cdot |T| \\
&\leq \exp(-\epsilon^2\mu t/2 - \mu \ln(1-\epsilon)) \cdot |T| \leq 1/\operatorname{poly}(|T|) < 1
\end{aligned}$$

Now we will show that $\Phi(t')$ does not increase as t' increase. Let $K(t') = \frac{e^{-\epsilon\mu(t-t')}}{(1-\epsilon)^{-(t/|A|)(t')+\mu(t-t'+1)-\epsilon\mu t}}$. Let $j^* = f_2(X_{t'}, a)$ be the task returned by the feedback function that is independent of a. Note that by definition of f_1 and f_2, for $t' < t$, we have:

$$\begin{aligned}
w^{(t'+1)} &\geq \lfloor w^{(t')} \cdot (t'/(t'+1)) \rfloor + [j = j^*] \cdot |A| \cdot (1/(t'+1)) \\
w^{(t'+1)} &\geq w^{(t')} \cdot (t'/(t'+1)) - 1 + [j = j^*] \cdot |A| \cdot (1/(t'+1)) \\
(t'+1) \cdot w^{(t'+1)} &\geq w^{(t')} \cdot t' - (t'+1) + [j = j^*] \cdot |A| \\
(t'+1) \cdot x^{(t'+1)} &\geq x^{(t')} \cdot t' - t/|A| + [j = j^*] \qquad (7.3)
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\Phi(t'+1) \\
&= K(t'+1) \cdot \sum_j \frac{(1-\epsilon)^{x_j^{(t'+1)} \cdot (t'+1)}}{(1-\epsilon)^{\mu_j t'}} \\
&\leq K(t'+1) \cdot \sum_j \frac{(1-\epsilon)^{x_j^{(t')} \cdot t' - t/|A|} \cdot (1-\epsilon)^{[j=j^*]}}{(1-\epsilon)^{\mu_j t'}} && \text{by (7.3)} \\
&= K(t'+1) \cdot \left(\sum_j (1-\epsilon)^{x_j^{(t')} \cdot t' - \mu_j t' - t/|A|} \cdot (1 - [j = j^*] \cdot \epsilon) \right) \\
&\leq K(t'+1) \cdot \left(\sum_j (1-\epsilon)^{x_j^{(t')} \cdot t' - \mu_j t' - t/|A|} \cdot (1 - \epsilon \cdot \mu_j) \right) && j^* = \max_j \mu_j t' - x_j^{(t')} t' \\
&= K(t'+1) \cdot \left(\sum_j (1-\epsilon)^{x_j^{(t')} \cdot t' - \mu_j t' - t/|A|} \cdot (1-\epsilon)^{\mu_j} \cdot \frac{(1 - \epsilon \cdot \mu_j)}{(1-\epsilon)^{\mu_j}} \right) \\
&\leq K(t'+1) \cdot \left(\sum_j (1-\epsilon)^{x_j^{(t')} \cdot t' - \mu_j (t'-1) - t/|A|} \cdot \frac{e^{-\epsilon\mu_j}}{(1-\epsilon)^{\mu_j}} \right) && 1 - x \leq e^{-x} \\
&\leq K(t'+1) \cdot \left(\sum_j (1-\epsilon)^{x_j^{(t')} \cdot t' - \mu_j (t'-1) - t/|A|} \cdot \frac{e^{-\epsilon\mu}}{(1-\epsilon)^{\mu}} \right) && \mu \geq \mu_j \text{ and } \tfrac{e^{-\epsilon}}{1-\epsilon} \geq 1 \\
&\leq K(t') \cdot \left(\sum_j \frac{(1-\epsilon)^{x_j^{(t')} \cdot t'}}{(1-\epsilon)^{\mu_j (t'-1)}} \right) \\
&= \Phi(t')
\end{aligned}$$

Therefore, $\Phi(t) \leq \Phi(t-1) \leq \ldots \leq \Phi(0) < 1$, hence

$$\begin{aligned}
1 &> \Phi(t) \\
&= \sum_j \frac{1}{(1-\epsilon)^{\mu - \epsilon\mu t - t^2/|A|}} \cdot \frac{(1-\epsilon)^{x_j^{(t)} \cdot t}}{(1-\epsilon)^{\mu_j (t-1)}} \\
&\geq \sum_j \frac{1}{(1-\epsilon)^{\mu_j - \epsilon\mu t - t^2/|A|}} \cdot \frac{(1-\epsilon)^{x_j^{(t)} \cdot t}}{(1-\epsilon)^{\mu_j (t-1)}} && \mu \geq \mu_j \\
&\geq \sum_j \frac{(1-\epsilon)^{x_j^{(t)} \cdot t}}{(1-\epsilon)^{\mu_j t - \epsilon\mu t - t^2/|A|}}
\end{aligned}$$

This implies that $x_j^{(t)} \geq \mu_j - \epsilon\mu - t/|A|$ for all j, which in turns implies $w_j^{(t)} \geq d_j - \epsilon d_{\max} - t/|A|$ for all j. Note that when $|A| = \sum_j d_j$, the time bound is $O(\epsilon^{-2}|T| \log |T|)$ since $|A|/d_{\max} \leq |T|$. □

7.4 Discussion and Future Work

We have presented a model that is capable of achieving an approximate optimal solution for the task allocation problem with individual variation. However, the error and the convergence time do not match that for the homogeneous and deterministic model. In particular, it would be interesting to see if the additive error can be as small as $\Theta(d_{\max} + \log |T|)$, which will match the bound achieved by the randomized rounding technique in the centralized setting. Another direction to pursue in is to consider a a fixed probability of switching tasks rather than a probability that decreases over time. The analysis of the resulting algorithm would be useful in the setting of demands varying over time. We also hope that the resulting convergence time is independent or insensitive to $|A|$ which would make it consistent with the conjecture in [146].

Part III

Matching

Chapter 8

A Scaling Algorithm for Maximum Weight Perfect Matching in General Graphs

8.1 Introduction

Graph matching problems have always played a pivotal role in theoretical computer science, from the development of linear programming duality theory [82, 92, 113], polyhedral combinatorics [41], social choice theory [62], and the definition of the complexity class **P** [42]. In 1965 Edmonds [41, 42] proved that on general (non-bipartite) graphs, both the maximum cardinality matching and maximum weight matching problems could be solved in polynomial time. Some of the subsequent work on general weighted graph matching focused on developing faster implementations of Edmonds' algorithm [52, 55, 57, 63, 101, 116] whereas others pursued alternative techniques such as cycle-canceling [27], weight-scaling [54, 61], or an algebraic approach using fast matrix multiplication [28].

Computing an optimum (maximum or minimum) matching is the bottleneck in many applications. For example, the only non-trivial part of Christofides' 3/2-approximate Metric

TSP algorithm [22] is to compute a minimum weight perfect matching on a certain submetric, represented as a complete graph. Edmonds and Johnson [43] reduced Kwan's [115] Chinese Postman problem, namely to find a shortest tour visiting all edges at least once, to weighted matching. Very recently Gabow, and Sankowski [58] proved that the complexity of computing undirected single-source shortest paths (with arbitrary weights, but no negative cycles) is no more than that of weighted matching. Faster matching algorithms therefore lead directly to faster algorithms for solving numerous combinatorial optimization problems.

In this chapter we present a new *scaling* algorithm for solving the weighted matching problem on general graphs. In Sections 8.1.1–8.1.3 we review matching terminology and the history of weighted matching algorithms; in Section 8.1.4 we summarize our contributions and survey the rest of the chapter.

8.1.1 Terminology

The input is a graph $G = (V, E, \hat{w})$ where $|V| = n, |E| = m$, and $\hat{w} : E \to \mathbb{R}$ assigns a real weight to each edge. A *matching* M is a set of vertex-disjoint edges. A vertex is *free* if it is not adjacent to an M edge. An *alternating path* is one whose edges alternate between M and $E \setminus M$. An alternating path P is *augmenting* if it begins and ends with free vertices, which implies that $M \oplus P \stackrel{\text{def}}{=} (M \cup P) \setminus (M \cap P)$ is also a matching and has one more edge. The weight of M is the sum of its individual edge weights: $\hat{w}(M) \stackrel{\text{def}}{=} \sum_{e \in M} \hat{w}(e)$. The *maximum cardinality matching* (MCM) and *maximum weight matching* (MWM) problems are to find a matching M maximizing $|M|$ and $\hat{w}(M)$, respectively. The *maximum weight perfect matching* (MWPM) problem is to find a perfect matching M (or, in general, one with maximum cardinality) maximizing $\hat{w}(M)$. In this chapter we assume that $\hat{w} : E \to \{0, \ldots, N\}$ assigns non-negative integer weights bounded by N. Assuming non-negative weights is without loss of generality since we can simply subtract $\min_{e \in E}\{\hat{w}(e)\}$ from every edge weight, which does not affect the relative weight of two perfect matchings. Moreover, the *minimum* weight perfect matching problem is reducible to MWPM, simply by substituting $-\hat{w}$ for $\hat{w}$. All MWPM algorithms can be used to solve MWM in the same time bound but the complexities of the two problems are not known to be identical.

8.1.2 Edmonds' Algorithm

Edmonds' MWPM algorithm begins with an empty matching M and consists of a sequence of *search* steps, each of which performs zero or more *dual adjustment*, *blossom shrinking*, and *blossom dissolution* steps until a tight augmenting path emerges or the search detects that $|M|$ is maximum. (Blossoms, duals, and tightness are reviewed in Section 8.2.) The overall running time is therefore $O(n \cdot \mathrm{Edm}(m, n))$, where $\mathrm{Edm}(m, n)$ is the cost of one search. On bipartite graphs Edmonds' search procedure encounters no blossoms: it resembles Kuhn's *Hungarian search*. Whereas a Hungarian search is essentially computing single-source shortest paths, and can therefore be implemented efficiently using Dijkstra's algorithm [33] with a Fibonacci heap [50], Edmonds' search requires sophisticated data structures to determine when blossoms must be shrunk and dissolved. Following [57, 63], Gabow [55] gave an algorithm for Edmonds' search running in $O(m + n \log n)$ time, that is, linear time whenever $m = \Omega(n \log n)$. Many, though not all, of the data structure problems become simpler on integer-weighted graphs when there is a known bound t on the number of dual adjustments, or equivalently, the weighted length of the shortest augmenting path. Gabow [54] showed that one of Edmonds' searches takes $O(t + \textsc{SplitFindmin}(m, n))$ time, where $\textsc{SplitFindmin}(m, n)$ is the complexity of a constrained priority queue-type problem called split-findmin, which is used to manage blossom dissolution. The best bound on $\textsc{SplitFindmin}(m, n)$ (for real-weighted inputs) is $O(n + m \log \alpha(m, n))$ [147], where α is the slowly growing inverse-Ackermann function. However, Thorup [162] showed that split-find on integer-weighted inputs can be implemented in optimal $O(n + m)$ time using Fredman and Willard's atomic heaps [51].

8.1.3 Scaling Algorithms

The problem with Edmonds' MWPM algorithm is that it finds augmenting paths one at a time, apparently dooming it to a running time of $\Omega(mn)$. Gabow [54] showed that this $\Omega(mn)$ barrier could be broken using the *scaling* technique of Edmonds and Karp [44]. The idea is to expose the edge weights one bit at a time. In the ith scale the goal is to compute an optimum matching with respect to the i most significant bits of $\hat{w}$. The reason this method is efficient is because an optimal solution at scale $i-1$ yields an additive $O(n)$ approximation

to an optimal solution at scale i, making the general weighted problem closely resemble a sequence of $\log N$ *unweighted* matching problems. Gabow's algorithms (for bipartite and general MWPM) ran in $O(mn^{3/4} \log N)$ time, which beat $O(mn)$ but still left much room for improvement. The $O(m\sqrt{n})$-time MCM algorithm of Micali and Vazirani [122, 169] provides a natural *de facto* lower bound on the performance of any scaling algorithm for MWPM.

Gabow and Tarjan [60, 61] loosened the requirement that the algorithm find an *exactly* optimal solution at each scale. Since an additive $O(n)$ error is already introduced just by exposing the ith bit of $\hat{w}$, it suffices if the output at the end of the $(i-1)$th scale was *itself* erroneous up to an additive $O(n)$. The Gabow-Tarjan MWPM algorithms [60, 61] run in $O(m\sqrt{n}\log(nN))$ time on bipartite graphs and $O(m\sqrt{n\alpha(n,m)\log n}\log(nN))$ time on general graphs. Over the years other researchers [38, 68, 138] developed alternative $O(m\sqrt{n}\log(nN))$-time algorithms for *bipartite* MWPM, though the $O(m\sqrt{n\alpha(n,m)\log n}\log(nN))$-time bound of for general graphs has resisted all improvement. It has seen no asymptotic speedup,[1] nor has it been simplified in either its design or analysis.

It is straightforward to reduce MWM to MWPM without increasing m, n, or N asymptotically, but no similarly efficient reduction in the reverse direction is known; see [36]. Indeed, there is some evidence that MWM may, in some situations, be slightly easier than MWPM. Kao, Lam, Sung, and Ting [93] gave two reductions from *bipartite* MWM to bipartite MCM. The first makes N black-box calls to an MCM algorithm whereas the second makes a *single* call to an MCM algorithm but on a graph with up to $O(Nn)$ vertices and $O(Nm)$ edges. Huang and Kavitha [84] and Pettie [148] proved that even on general graphs, MWM could be solved with N calls to an MCM algorithm. In Chapter 9, we gave a scaling algorithm for bipartite MWM running in $O(m\sqrt{n}\log N)$ time, which is asymptotically faster than [60] whenever $\log N = o(\log(nN))$.

[1] (aside from those implied by better split-findmin technology. The $\alpha(m,n)$ here reflects Gabow's split-findmin structure [54]. It can be reduced to $\log \alpha(m,n)$ [147]. Thorup [162] noted that when the input consists of integers (rather than reals) the split-findmin data structure can perform all operations in amortized $O(1)$ time using Fredman and Willard's atomic heaps [51], thereby yielding an $O(m\sqrt{n\log n}\log(nN))$-time implementation of the Gabow-Tarjan MWPM algorithm.

Table 2

MAXIMUM WEIGHT PERFECT MATCHING ON GENERAL GRAPHS

Year	Authors	Time Complexity & Notes
1965	Edmonds	mn^2
1974 1976	Gabow Lawler	n^3
1976	Karzanov	$n^3 + mn \log n$
1978	Cunningham & Marsh	$\text{poly}(n)$
1982	Galil, Micali & Gabow	$mn \log n$
1985	Gabow	$mn^{3/4} \log N$ INTEGER WEIGHTS
1989	Gabow, Galil & Spencer	$mn \log\log\log_d n + n^2 \log n$ $d = 2 + m/n$
1990	Gabow	$mn + n^2 \log n$
1991	Gabow & Tarjan	$m\sqrt{n\alpha(n,m)\log n}\log(nN)$ INTEGER WEIGHTS
2012	Cygan, Gabow & Sankowski	Nn^ω RANDOMIZED, INTEGER WEIGHTS
new		$\text{Edm}(m,n,N) \cdot \sqrt{n}\log(nN)$ INTEGER WEIGHTS $m\sqrt{n}\log(nN)$

MAXIMUM WEIGHT MATCHING ON BIPARTITE AND GENERAL GRAPHS

Year	Authors	Time Complexity of $\text{MWM}(m,n,N)$
	folklore/trivial	$\text{MWPM}(2m+n, 2n, N)$
1999	Kao, Lam, Sung & Ting	$\text{MCM}(Nn, Nm)$ BIPARTITE, INTEGER WEIGHTS $N\cdot\ \text{MCM}(m,n)$ BIPARTITE, INTEGER WEIGHTS $\rightarrow\ Nm\sqrt{n}/\kappa$ $\kappa = \frac{\log n}{\log(n^2/m)}$, [49] $\rightarrow\ Nn^\omega$ RANDOMIZED, [130] $\rightarrow\ Nm^{10/7}\cdot\text{polylog}(n)$ RANDOMIZED, [123]
2012 2012	Huang & Kavitha Pettie	$N\cdot\ \text{MCM}(m,n)$ INTEGER WEIGHTS $\rightarrow\ Nm\sqrt{n}/\kappa$ $\kappa = \frac{\log n}{\log(n^2/m)}$, [67] $\rightarrow\ Nn^\omega$ RANDOMIZED, [79, 130]
	Chapter 9	$m\sqrt{n}\log N$ BIPARTITE, INTEGER WEIGHTS

8.1.4 New Results

We develop a new scaling approach for solving MWPM in general graphs that circumvents some inefficiencies and conceptual difficulties in Gabow and Tarjan's algorithm [61]. By synthesizing ideas from Gabow's original $O(mn^{3/4} \log N)$-time scaling algorithm for MWPM and a liquidationist approach, our algorithm for MWPM runs in $O(m\sqrt{n}\log(nN))$ time for *all* m, n, and N. The algorithm is not exactly simple, but its analysis and proof of correctness are entirely elementary.

Organization In Section 8.2 we give a reasonably detailed technical tour of our algorithm, which covers Edmonds' LP formulation of MWPM, the structure of *blossoms*, the scaling technique of Gabow and Gabow-Tarjan, and the difficulties of improving the Gabow-Tarjan algorithm. in Section 8.3, we present our HYBRID MWPM algorithm running in $O(m\sqrt{n}\log(nN))$. We conclude with some open problems in Section 8.4.

8.2 A Brief History of Matching Algorithms

8.2.1 Odd Set Constraints and Blossoms

The MWPM problem can be expressed as an integer linear program

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x(e) \cdot \hat{w}(e) \\ \text{subject to} \quad & x(e) \in \{0,1\}, \text{ for all } e \in E \\ \text{and} \quad & \sum_{e \supset v} x(e) = 1, \text{ for all } v \in V. \end{aligned}$$

The integrality constraint lets us interpret $\mathbf{x}$ as the membership vector of a set of edges and the $\sum_{e \supset v} x(e) = 1$ constraint enforces that $\mathbf{x}$ represents a perfect matching. Birkhoff's theorem [13] (see also von Neumann [171]) implies that in bipartite graphs the integrality constraint can be relaxed to $x(e) \in [0,1]$. The basic feasible solutions to the resulting LP correspond to perfect matchings. However, this is not true of non-bipartite graphs! Edmonds proposed exchanging the integrality constraint for an exponential number of the following

odd set constraints, which are obviously satisfied for every $\mathbf{x}$ that is the membership vector of a matching.

$$\sum_{e \subset B} x(e) \leq \lfloor |B|/2 \rfloor, \text{ for all } B \subset V, |B| \geq 3 \text{ odd}.$$

Edmonds proved that the basic feasible solutions to the resulting LP are integral and therefore correspond to perfect matchings. Weighted matching algorithms work directly with the dual LP. Let $y : V \to \mathbb{R}$ and $z : 2^V \to \mathbb{R}$ be the vertex duals and odd set duals.

$$\begin{aligned}
\text{minimize} \quad & \sum_{v \in V} y(v) + \sum_{\substack{B \subset V: \\ |B| \geq 3 \text{ is odd}}} z(B) \cdot \lfloor |B|/2 \rfloor \\
\text{subject to} \quad & z(B) \geq 0, \text{ for all odd } B \subset V, \\
& \hat{w}(u,v) \leq yz(u,v) \text{ for all } (u,v) \in E, \\
\text{where, by definition,} \quad & yz(u,v) \stackrel{\text{def}}{=} y(u) + y(v) + \sum_{B \supset \{u,v\}} z(B).
\end{aligned}$$

We generalize the yz dual function to an arbitrary set $S \subseteq V$ of vertices as follows.

$$yz(S) = \sum_{u \in S} y(u) \; + \; \sum_{B \subseteq S} z(B) \cdot \lfloor |B|/2 \rfloor \; + \; \sum_{S \subset B} z(B) \cdot \lfloor |S|/2 \rfloor.$$

Note that $yz(V)$ is exactly the dual objective function.

Edmonds' algorithm [41, 42] maintains a dynamic matching M and dynamic laminar set $\Omega \subset 2^V$ of odd sets, each associated with a *blossom* subgraph. Informally, a blossom is an odd-length alternating cycle (w.r.t. M), whose constituents are either individual vertices or blossoms in their own right. More formally, blossoms are constructed inductively as follows. If $v \in V$ then the odd set $\{v\}$ induces a trivial blossom with edge set $\emptyset$. Suppose that for some even $\ell \geq 2$, $A_0, \ldots, A_\ell$ are disjoint odd sets associated with blossoms $E_{A_0}, \ldots, E_{A_\ell}$. If there are edges $e_0, \ldots, e_\ell \in E$ such that $e_i \in A_i \times A_{i+1}$ (modulo $\ell + 1$) and $e_i \in M$ if and only if i is odd, then $B = \bigcup_i A_i$ is an odd set associated with the blossom $E_B = \bigcup_i E_{A_i} \cup \{e_0, \ldots, e_\ell\}$. Because ℓ is even, the alternating cycle on $A_0, \ldots, A_\ell$ has odd length, leaving A_0 incident to two unmatched edges, e_0 and e_ℓ. One can easily prove by induction that $|B|$ is odd and that $E_B \cap M$ matches all but one vertex in B, called the *base* of B. The base of B is the

same as the base of A_0. Remember that $E(B) = E \cap \binom{B}{2}$, the edge set induced by B, may contain many non-blossom edges not in E_B. Define $n(B) = |B|$ and $m(B) = |E(B)|$ to be the number of vertices and edges in the graph induced by B.

The set Ω of *active* blossoms is represented by rooted trees, where leaves represent vertices and internal nodes represent nontrivial blossoms. A *root blossom* is one not contained in any other blossom. The children of an internal node representing a blossom B are ordered by the odd cycle that formed B, where the child containing the base of B is ordered first. Edmonds [41, 42] showed that it is often possible to treat blossoms as if they were single vertices, by *shrinking* them. We obtain the *shrunken graph* G/Ω by contracting all root blossoms and removing the edges in those blossoms. To *dissolve* a root blossom B means to delete its node in the blossom forest and, in the contracted graph, to replace B with individual vertices $A_0, \ldots, A_\ell$.

Blossoms have numerous properties. Our algorithms use two in particular.

1. The subgraph on E_B is *critical*, meaning it contains a perfect matching on $B \backslash \{v\}$, for each $v \in B$. Phrased differently, any $v \in B$ can be made the base of B by choosing the matching edges in E_B appropriately.

2. As a consequence of (1), any augmenting path P' in G/Ω extends to an augmenting path P in G, by replacing each non-trivial blossom vertex B in P' with a corresponding path through E_B. Moreover, Ω is still valid for the matching $M \oplus P$, though the bases of blossoms intersecting P may be relocated by augmenting along P. See Figure 4 for an example.

8.2.2 Relaxed Complementary Slackness

Edmonds' algorithm maintains a matching M, a nested set Ω of blossoms, and duals $y : V \to \mathbb{Z}$ and $z : 2^V \to \mathbb{N}$ that satisfy Property 8.2.1. Here w is a weight function assigning *even* integers; it is usually not the same as the input weights $\hat{w}$.

Property 8.2.1. *(Complementary Slackness)*

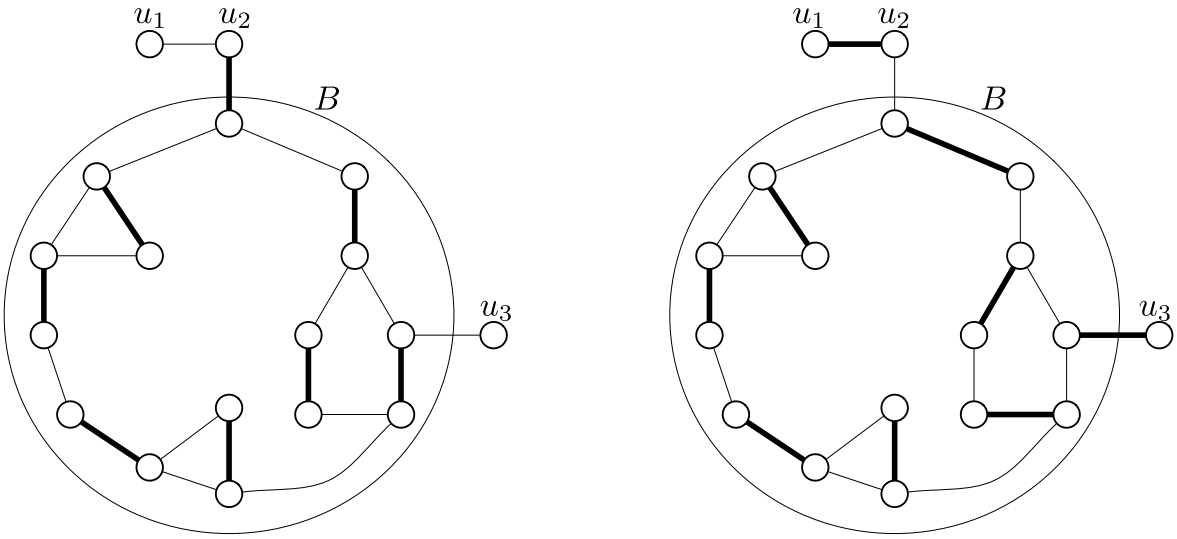


Figure 4: Matched edges are thick, unmatched edges thin. Left: B is a blossom consisting of 7 sub-blossoms, 4 of which are trivial (vertices) and the other three non-trivial blossoms. The path $P' = (u_1, u_2, B, u_3)$ is an augmenting path in the shrunken graph $G/\{B\}$. Right: augmenting along P' in $G/\{B\}$ enlarges the matching and has the effect of moving the base of B to the vertex matched with u_3.

1. *Granularity.* $z(B)$ *is a nonnegative even integer for each* $B \in \Omega$*, and* $y(u)$ *is an integer for each vertex* u*.*

2. *Active Blossoms.* $|M \cap E_B| = \lfloor |B|/2 \rfloor$ *for all* $B \in \Omega$*. If* $B \in \Omega$ *is a root blossom then* $z(B) > 0$*; if* $B \notin \Omega$ *then* $z(B) = 0$*. Non-root blossoms may have zero* z*-values.*

3. *Domination.* $yz(e) \geq w(e)$*, for each* $e = (u, v) \in E$*.*

4. *Tightness.* $yz(e) = w(e)$*, for each* $e \in M \cup \bigcup_{B \in \Omega} E_B$*.*

Lemma 8.2.2. *If Property 8.2.1 is satisfied for a perfect matching* M*, blossom set* Ω*, and duals* y, z*, then* M *is necessarily a* MWPM *w.r.t. the weight function* w*.*

The proof of Lemma 8.2.2 follows the same lines as Lemma 8.2.4, proved below.

The Gabow-Tarjan algorithms and their successors [36,38,60,61,68,138] maintain a relaxation of complementary slackness. By using Property 8.2.3 in lieu of Property 8.2.1 we introduce an additive error as large as n. This does not prevent us from computing exact MWPMs but it does necessitate additional scales. Before the algorithm proper begins we compute the extended weight function $\bar{w}(e) = (n+1)\hat{w}(e)$. Because the weight of every matching w.r.t. $\bar{w}$ is a multiple of $n + 1$, if $\bar{w}(M)$ is provably within n of optimal then M must be exactly optimal w.r.t. $\bar{w}$ and hence $\hat{w}$ as well.

Property 8.2.3. *(Relaxed Complementary Slackness) Property 8.2.1(1,2) holds and Property 8.2.1(3,4) are replaced with*

3. *Near Domination.* $yz(e) \geq w(e) - 2$ *for each edge* $e = (u, v) \in E$*.*

4. *Near Tightness.* $yz(e) \leq w(e)$*, for each* $e \in M \cup \bigcup_{B \in \Omega} E_B$*.*

The next lemma shows that a perfect matching satisfying Property 8.2.3 will be a good approximation of the maximum weight perfect matching.

Lemma 8.2.4. *If Property 8.2.3 is satisfied for some perfect matching* M*, blossom set* Ω*, and duals* y, z*, then* $w(M) \geq w(M^*) - n$*.*

Proof. By Property 8.2.3 (near tightness and active blossoms) and the definition of yz,

$$w(M) \geq \sum_{e \in M} yz(e) = \sum_{u \in V} y(u) + \sum_{B \in \Omega} z(B) \cdot |M \cap E(B)| = \sum_{u \in V} y(u) + \sum_{B \in \Omega} z(B) \cdot \lfloor |B|/2 \rfloor.$$

Since the MWPM M^* puts at most $\lfloor |B|/2 \rfloor$ edges in any blossom $B \in \Omega$,

$$\begin{aligned} w(M^*) &\leq \sum_{u \in V} y(u) + \sum_{B \in \Omega} z(B) \cdot |M^* \cap E(B)| + 2|M^*| && \text{Property 8.2.3 (near domination)} \\ &\leq \sum_{u \in V} y(u) + \sum_{B \in \Omega} z(B) \cdot \lfloor |B|/2 \rfloor + n. && |M^*| = n/2, \text{ Non-negativity of } z \end{aligned}$$

Therefore, we have $w(M) \geq w(M^*) - n$. □

8.2.3 Edmonds' Search

Suppose we have a matching M, blossom set Ω, and duals y, z satisfying Property 8.2.1 or 8.2.3. The goal of Edmonds' *search* procedure is to manipulate y, z, and Ω until an *eligible* augmenting path emerges. At this point $|M|$ can be increased by augmenting along such a path (or multiple such paths), which preserves Property 8.2.1 or 8.2.3. The definition of *eligible* needs to be compatible with the governing invariant (Property 8.2.1 or 8.2.3) and other needs of the algorithm. In our algorithms we use several implementations of Edmonds' generic search: they differ in their governing invariants, definition of eligibility, and data structural details. For the time being the reader can assume Property 8.2.1 is in effect and that we use Edmonds' original eligibility criterion [41].

Criterion 1. *An edge e is* eligible *if it is* tight*, that is, $yz(e) = w(e)$. (If Property 8.2.1 is satisfied then all matched and blossom edges are tight.)*

Let E_{elig} be the set of eligible edges and $G_{\text{elig}} = (V, E_{\text{elig}})/\Omega$ be the graph obtained by contracting all root blossoms and discarding ineligible edges. We consider a slight variant of Edmonds' search that looks for augmenting paths only from a specified set F of free vertices, that is, each augmenting path must have at least one end in F and possibly both. The search iteratively performs *Augmentation*, *Blossom Shrinking*, *Dual Adjustment*, and

Blossom Dissolution steps, halting after the first Augmentation step that discovers at least one augmenting path. We require that the y-values of all F vertices have the same parity (even/odd). This is needed to keep y, z integral and allow us to perform discrete dual adjustment steps without violating Property 8.2.1 or 8.2.3. See Figure 5 for the pseudocode.

There is considerable flexibility in implementing Edmonds' search. If the edge weights are unbounded then we have no *a priori* upper bound on the number of dual adjustments needed to find an augmenting path. In such situations Edmonds' search is implemented with the assistance of a special priority queue that keeps track of which dual adjustment steps induce a change in the structure of G_{elig}. This can be the dissolution of a blossom or the creation of a new eligible edge, which, in turn, could create a new blossom or augmenting path. We use PQSearch to refer to the best implementation of Edmonds' search with such a priority queue, which can handle an *unbounded* number of dual adjustments. When there are at most $t = O(m)$ dual adjustments we can implement the underlying priority queue with an array of t *buckets*, which support insert, deletemin, and decreasekey in optimal constant time. We refer to this implementation of Edmonds' algorithm as BucketSearch. Aside from the priority queue, all other non-trivial data structures can be implemented to run in linear time [54, 59, 162].

Regardless of what t is or how the dual adjustments are handled, we still have options for how to implement the *Augmentation* step. Under Criterion 1 of eligibility, the Augmentation step always extends M to a maximum cardinality matching in the subgraph of G_{elig} induced by $V(M) \cup F$. This can be done in $O((p+1)m)$ time if $p \geq 0$ augmenting paths are found [59], or in $O(m\sqrt{n})$ time, independent of p, using the Micali-Vazirani algorithm [122, 169] or Gabow-Tarjan cardinality matching algorithm [61, §10].

8.2.4 Relaxed Complementary Slackness

Each scale of the Gabow-Tarjan MWPM algorithm maintains Property 8.2.3 (Relaxed Complementary Slackness) which requires a correspondingly relaxed criterion for *eligibility*. Roughly speaking, an edge is eligible if the inequalities of Property 8.2.3(near domination, near tightness) hold with equality.

Criterion 2. *An edge e is* eligible *in if at least one of the following holds.*

EDMONDSSEARCH(F) *Precondition:* $\{y(u) \mid u \in F\}$ *must all be of the same parity.*

Repeatedly perform *Augmentation*, *Blossom Shrinking*, *Dual Adjustment*, and *Blossom Dissolution* steps until a halting condition is reached.

- *Augmentation:*
  While G_{elig} contains an augmenting path from some free vertex in F, find such a path P and set $M \leftarrow M \oplus P$. Update G_{elig}.

- *Blossom Shrinking:*
  Let $V_{out} \subseteq V(G_{\text{elig}})$ be the vertices (that is, root blossoms) reachable from free vertices in F by even-length alternating paths; let Ω_{new} be a maximal set of (nested) blossoms on V_{out}. (That is, if $(u, v) \in E(G_{\text{elig}})\backslash M$ and $u, v \in V_{out}$, then u and v must be in a common blossom in Ω_{new}.) Let $V_{in} \subseteq V(G_{\text{elig}})\backslash V_{out}$ be those vertices reachable free vertices in F by odd-length alternating paths. Set $z(B) \leftarrow 0$ for $B \in \Omega_{new}$ and set $\Omega \leftarrow \Omega \cup \Omega_{new}$. Update G_{elig}.

- *Dual Adjustment:*
  Let $\hat{V}_{in}, \hat{V}_{out} \subseteq V$ be original vertices represented by vertices in V_{in} and V_{out}. The y- and z-values for some vertices and root blossoms are adjusted:

  $$y(u) \leftarrow y(u) - 1, \text{ for all } u \in \hat{V}_{out}.$$
  $$y(u) \leftarrow y(u) + 1, \text{ for all } u \in \hat{V}_{in}.$$
  $$z(B) \leftarrow z(B) + 2, \text{ if } B \in \Omega \text{ is a root blossom with } B \subseteq \hat{V}_{out}.$$
  $$z(B) \leftarrow z(B) - 2, \text{ if } B \in \Omega \text{ is a root blossom with } B \subseteq \hat{V}_{in}.$$

- *Blossom Dissolution:*
  After dual adjustments some root blossoms may have zero z-values. Dissolve such blossoms (remove them from Ω) as long as they exist. Update G_{elig}.

Figure 5: A generic implementation of Edmonds' search procedure. Data structural issues are ignored, as is the eligibility criterion, which determines G_{elig}.

Search Procedure	Invariants and Eligibility	Time Bound and Data Structures	
PQSearch	Property 8.2.1, Criterion 1 Property 8.2.3, Criterion 3	$O((p+1)m + n\log n)$	[55, 59]
BucketSearch	Property 8.2.3, Criterion 3	$O(t + (p+1)m)$	[54, 59, 162]
SearchOne	Property 8.2.3, Criterion 2	$O(m)$	[59, 61, 162]
ShellSearch	Props. 8.2.1, 8.2.8, Criterion 1	$O(t + m \cdot \min\{\sqrt{n}, p+1\})$	[54, 122, 169]

Table 3: Here m and n are the number of edges and vertices involved in the search, $p \geq 0$ is the number of augmenting paths discovered in the final augmentation step, and t the number of dual adjustments performed. The running time of PQSearch is independent of t. The running time of SearchOne is independent of p; it can only be applied for $t = 1$ dual adjustment.

1. $e \in E_B$ *for some* $B \in \Omega$.
2. $e \notin M$ *and* $yz(e) = w(e) - 2$.
3. $e \in M$ *and* $yz(e) = w(e)$.

Criterion 2(1) guarantees that eligible augmenting paths in the shrunken graph $G_{\text{elig}} = (V, E_{\text{elig}})/\Omega$ correspond to eligible augmenting paths in G. A key consequence of Criterion 2(2,3) is that augmenting along an eligible augmenting path P in G_{elig} *makes all edges in* P *ineligible*. Thus, an efficient implementation of the *Augmentation* step need only find a maxi*mal* set of augmenting paths from F, not a maxi*mum* set of such paths.[2] The procedure SearchOne(F) is used only in conjunction with Property 8.2.3 and Criterion 2 of eligibility. It performs precisely *one* dual adjustment step and might not find any augmenting paths. See Figure 6.

Lemma 8.2.5. *After the Augmentation step of* SearchOne(F) *(using Criterion 2 for eligibility),* G_{elig} *contains no eligible augmenting paths from an* F*-vertex.*

Proof. Suppose that, after the Augmentation step, there is an augmenting path P from an F-vertex in G_{elig}. Since Ψ was maximal, P must intersect some $P' \in \Psi$ at a vertex v. However, after the Augmentation step every edge in P' will become ineligible, so the matching edge $(v, v') \in M$ is no longer in G_{elig}, contradicting the fact that P consists of eligible edges. □

[2] In the context of flow algorithms [71], this distinction is analogous to the difference between blocking flows and maximum flows.

SEARCHONE(F) *Precondition:* $\{y(u) \mid u \in F\}$ *must all be of the same parity.*

- *Augmentation:*
  Find a maximal set Ψ of vertex-disjoint augmenting paths from F in G_{elig}.
  Set $M \leftarrow M \oplus (\bigcup_{P \in \Psi} P)$.
- Perform *Blossom Shrinking* and *Dual Adjustment* steps from F, then *Blossom Dissolution*, exactly as in EDMONDSSEARCH.

Figure 6

It is possible to efficiently execute PQSEARCH/BUCKETSEARCH while maintaining Property 8.2.3 but, like Gabow and Tarjan [61], we need to introduce a third eligibility criterion to deal with a subtle issue arising from dissolved blossoms. When a blossom B is formed under Criterion 2 its matched edges are tight and its unmatched $e \in E_B$ have $yz(e) = w(e) - 2$. If B (as a shrunken vertex) participates in an augmenting path the matched/unmatched status of some edges in E_B will be reversed. If, after a subsequent dual adjustment step, B is dissolved, the reversed edges in E_B will no longer be eligible according to Criterion 2. However, implementations of PQSEARCH/BUCKETSEARCH rely on the fact that the subgraph of G reachable from F grows monotonically as dual adjustments are performed. In particular, if P is an eligible alternating path in G from some vertex in F a dual adjustment cannot make P ineligible. Thus, to use PQSEARCH/BUCKETSEARCH with Property 8.2.3 we use Criterion 3 of eligibility.

Criterion 3. *An edge is* eligible *if* $yz(e) = w(e)$ *or* $yz(e) = w(e) - 2$.

8.2.5 EDMONDSSEARCH and Properties 8.2.1 and 8.2.3

Edmonds' original algorithm searches from a *single* free vertex and does the maximum amount of dual adjustment that does not violate Property 8.2.1. Our variant differs in a few minor respects. We search from a specified *set* F of free vertices. We will show that Property 8.2.1 or 8.2.3 is maintained by EDMONDSSEARCH(F), assuming that w assigns only even weights and that the y-values of F-vertices have the same parity.

Lemma 8.2.6. *If Property 8.2.1 is satisfied and the y-values of vertices in F have the same*

parity, then Property 8.2.1 remains satisfied after EdmondsSearch(F) *under Criterion 1 for eligibility.*

Proof. Property 8.2.1 (granularity) is obviously maintained, since we are always adjusting y-values by 1 and z-values by 2. Property 8.2.1 (active blossoms) is also maintained since all the new root blossoms discovered in the Blossom Shrinking step are in V_{out} and will have positive z-values after adjustment. Furthermore, each root blossom whose z-value drops to zero is removed.

Consider the tightness and the domination conditions of Property 8.2.1. First note that if both endpoints of e lie in the same blossom, $yz(e)$ will not change until the blossom is dissolved. When the blossom was formed, the blossom edges must be eligible (tight). The augmentation step only makes eligible edges matched, so tightness is satisfied.

Consider the effect of a dual adjustment on an edge $e = (u, v)$, whose endpoints lie in different blossoms. We divide the analysis into the following four cases. Refer to Figure 7 for illustrations of the cases.

1. Both u and v are in $\hat{V}_{in} \cup \hat{V}_{out}$ and $e \in M$. We cannot have both $u, v \in \hat{V}_{out}$ (otherwise they would be in a common blossom, since e is eligible) nor can both be in $\hat{V}_{in}$, so $u \in \hat{V}_{in}, v \in \hat{V}_{out}$ and $yz(e)$ is unchanged.

2. Both u and v are in $\hat{V}_{in} \cup \hat{V}_{out}$ and $e \notin M$. If at least one of u or v is in $\hat{V}_{in}$, then $yz(e)$ cannot decrease and domination holds. Otherwise we must have $u, v \in \hat{V}_{out}$. In this case, e must be ineligible, for otherwise an augmenting path or a blossom would have been found. Ineligibility implies $yz(e) \geq w(e) + 1$ but something stronger can be inferred. Since the y-values of free vertices have the same parity, all vertices reachable from free vertices by eligible alternating paths also have the same parity. Since $w(e)$ is even (by assumption) and $yz(e)$ is even (by parity) we can conclude that $yz(e) \geq w(e) + 2$ before dual adjustment, and therefore $yz(e) \geq w(e)$ after dual adjustment.

3. u but not v is in $\hat{V}_{in} \cup \hat{V}_{out}$ and $e \in M$. This case cannot happen since in this case, $u \in \hat{V}_{in}$ and e must be ineligible, but we know all matched edges are tight.

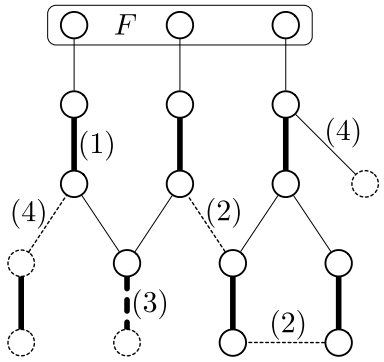


Figure 7: Thick edges are matched, thin unmatched. Dashed edges (whether thick or thin) are ineligible. Solid vertices are in $\hat{V}_{in} \cup \hat{V}_{out}$; all other vertices are dashed. Case (3) can only occur under Criteria 2 or 3 of eligibility.

4. u but not v is in $\hat{V}_{in} \cup \hat{V}_{out}$ and $e \notin M$. If $u \in \hat{V}_{in}$, then $yz(e)$ increases and domination holds. Otherwise, $u \in \hat{V}_{out}$ and e must be ineligible. In this case, we have $yz(e) \geq w(e) + 1$ before the dual adjustment and $yz(e) \geq w(e)$ afterwards.

□

Lemma 8.2.7. *If Property 8.2.3 is satisfied and the y-values of vertices in F have the same parity, then Property 8.2.3 remains satisfied after* EdmondsSearch(F) *under Criteria 2 or 3 for eligibility.*

Proof. The proof is similar to that of the previous lemma, except that we replace the tightness and domination by near tightness and near domination. We point out the differences in the following. An edge e can be included in a blossom only if it is eligible. An eligible edge must have $yz(e) = w(e)$ or $yz(e) = w(e) - 2$. Augmentations only make eligible edges matched. Therefore near tightness is satisfied after the Augmentation step.

When doing the dual adjustment, the following are the cases when $yz(e)$ is modified after the dual adjustment. In Case 2 of the previous proof, when $u, v \in \hat{V}_{out}$ but e is ineligible we must, by parity, have $yz(e) \geq w(e)$ before the dual adjustment and $yz(e) \geq w(e) - 2$ afterwards. Case 3 may happen in this situation. It is possible that $u \in \hat{V}_{in}$ and $e \in M$ is ineligible. Then we must have $yz(e) \leq w(e) - 1$ before the dual adjustment and $yz(e) \leq w(e)$

afterwards. In Case 4, when $u \in \hat{V}_{out}$, we have $yz(e) \geq w(e) - 1$ before the dual adjustment and $yz(e) \geq w(e) - 2$ afterwards. □

8.2.6 The Scaling Routine of Gabow and Gabow-Tarjan

The $(i-1)$th scale of the Gabow-Tarjan algorithm [61] ends when we have a perfect matching M' satisfying Property 8.2.3 with respect to some y', z', Ω', and weight function w'. Here $w'(e)$ consists of the $i-1$ high order bits of $\bar{w}(e)$, with a trailing zero bit to ensure that it is even. By Lemma 8.2.4 $w'(M')$ and $y'z'(V)$ differ by at most n. To begin the ith scale the Gabow-Tarjan algorithm reveals the ith bit of each edge weight, yielding w, and sets $y \leftarrow 2y' + 2, z \leftarrow 2z$. Now $w(M')$ and $yz(V)$ differ by $O(n)$ but M' is no longer a good matching since it does not satisfy Property 8.2.3. When the graph is bipartite Gabow and Tarjan [60] can simply discard M'—it is the y-values that are important—and start afresh at scale i with an empty matching. In general graphs, however, one can discard M' but getting rid of the old blossom structure Ω' is more problematic. Gabow [54] gave a method to dismantle old blossoms while maintaining Property 8.2.1, which led to an $O(mn^{3/4} \log N)$-time algorithm. Gabow and Tarjan's method [60] is based on Property 8.2.3, which is more efficient but whose analysis is more complex by orders of magnitude.

Gabow [54] noted that old blossoms could be eliminated via *translations*, which preserve (near) domination. Translating $B \in \Omega'$ by an integer Δ means setting

$$\begin{aligned} z(B) &\leftarrow z(B) - 2\Delta, && \text{assuming } z(B) \geq 2\Delta, \\ y(u) &\leftarrow y(u) + \Delta, && \text{for all } u \in B. \end{aligned}$$

From a global perspective, a translation by Δ increases the dual objective $yz(V)$ by $\Delta = -\lfloor |B|/2 \rfloor \cdot 2\Delta + |B| \cdot \Delta$. From a local perspective, a translation has no effect on $yz(e)$ if e has both or neither endpoint in B; it increases it by Δ if e has one endpoint in B. Thus, blossom translations are compatible with Property 8.2.1 or Property 8.2.3 so long as no matched edge in the ith scale straddles an undissolved blossom in Ω'. If some $e \in M$ did straddle $B \in \Omega'$, translating B could violate (near) tightness. After discarding the matching at the beginning of a scale, the algorithms of Gabow [54] and Gabow-Tarjan [61] fail to satisfy the full blossom criterion of Property 8.2.1(2) or Property 8.2.3(2), that is, that $z(B) > 0$

implies $|E(B) \cap M| = \lfloor |B|/2 \rfloor$. The algorithms of [54, 61] satisfy Property 8.2.8 in lieu of Property 8.2.1/8.2.3(2).

Property 8.2.8. *Let Ω' denote the set of as-yet undissolved blossoms from the previous scale and Ω, M be the blossom set and matching from the current scale.*

1. *$\Omega' \cup \Omega$ is a laminar (hierarchically nested) set.*
2. *There do not exist $B \in \Omega, B' \in \Omega'$ with $B' \subseteq B$.*
3. *No $e \in M$ has exactly one endpoint in some $B' \in \Omega'$.*
4. *If $B \in \Omega$ and $z(B) > 0$ then $|E_B \cap M| = \lfloor |B|/2 \rfloor$. An Ω-blossom is called a* root blossom *if it is not contained in any other Ω-blossom. All root blossoms have positive z-values.*

We have the option to *liquidate* all existing blossoms—just translate each $B \in \Omega'$ by $z(B)/2$—but it is impossible to place any useful upper bound on the dual objective $yz(V)$ afterward. Remember that the efficiency of scaling algorithms depends on the fact that $yz(V)$ is an additive $O(n)$ approximation of $w(M^*)$.

Gabow [54] observed that the two high level goals of a scaling MWPM algorithm (dismantling the old Ω' at scale $i-1$ and generating a new M, Ω at scale i) are actually not at odds but provide *mutual* support for each other. Dual adjustments (performed during Edmonds' searches) *reduce* $yz(V)$ and can help offset increases in $yz(V)$ caused by blossom translations. On the other hand, blossom translations help to enforce Property 8.2.8(3) by preventing edges straddling undissolved Ω' blossoms from violating domination. By correctly interleaving Edmonds' searches and old blossom translations, both goals can be achieved efficiently, without causing great distortions to the dual objective $yz(V)$. Below we explain Gabow's framework [54] for dismantling old blossoms, modified versions of which are used in [61] and in one of our MWPM algorithms.

The old blossom set Ω' is represented as a forest of rooted trees, with vertices at the leaves. Gabow [54] proposed to dismantle Ω' according to a *major path* decomposition of the blossom forest. A subblossom B' of B is called a *major child* if $|B'| > |B|/2$. Clearly each blossom has at most one major child. A blossom B that is not the major child of its parent is the root

of a *major path*. The major path of B is obtained by starting at B, iteratively moving from the current node to its major child, so long as it exists. The algorithms of [54, 61] dismantle Ω' by visiting each blossom B in postorder, calling DismantlePath(B) if B is the root of a major path. The DismantlePath(B) procedure ultimately dissolves B and every old blossom contained in B, and may along the way alter the matching M, the new blossom set Ω, and duals y, z.

When DismantlePath(B) is first called the major path rooted at B consists of undissolved blossoms $B_1 \subset B_2 \subset \cdots \subset B_k = B$ with positive z-values. The graph induced by $B_j \backslash B_{j-1}$ is called a *shell*, denoted $G(B_j, B_{j-1})$. All shells contain an even number of vertices, except for $G(B_1, \emptyset)$. Imagine executing some implementation of EdmondsSearch(F) on the free vertices F of some shell $G(C, D)$. Each dual adjustment step decrements the y-values of vertices in $\hat{V}_{out}$, which could cause a straddling edge (u, v) to violate (near) domination, where either

- $u \in C \backslash D$ and $v \in D$, or
- $u \in C \backslash D$ and $v \notin C$.

By translating C by 1 and D by 1 we increase $yz(u, v)$ for both types of straddling edges, thereby preserving Property 8.2.3(near domination). Edges straddling undissolved Ω' blossoms are automatically regarded as ineligible, which preserves Property 8.2.8(3). When the z-value of an Ω' blossom becomes zero it is dissolved, which has the effect of merging two adjacent shells.

The scaling algorithms of [54, 61] differ in numerous details, but both consist of rounds, each of which involves performing some number of steps of Edmonds' search on each remaining shell. Progress is made by either reducing $yz(B)$, matching vertices by finding augmenting paths, or dissolving Ω' blossoms. Eventually one of two events occurs: either (i) B and all its old subblossoms dissolve (due to translations), or (ii) there is exactly one free vertex inside the largest as-yet undissolved blossom. If (i) occurs the DismantlePath(B) routine halts immediately. If (ii) occurs a single Edmonds' search is conducted from v, which ultimately gets rid of each old blossom, either by dissolving it or making it a new blossom.

The analysis of DismantlePath(B) vaguely resembles the blocking-flow type analyses of cardinality matching algorithms [83, 100, 122, 169]. Gabow's original algorithm [54] uses

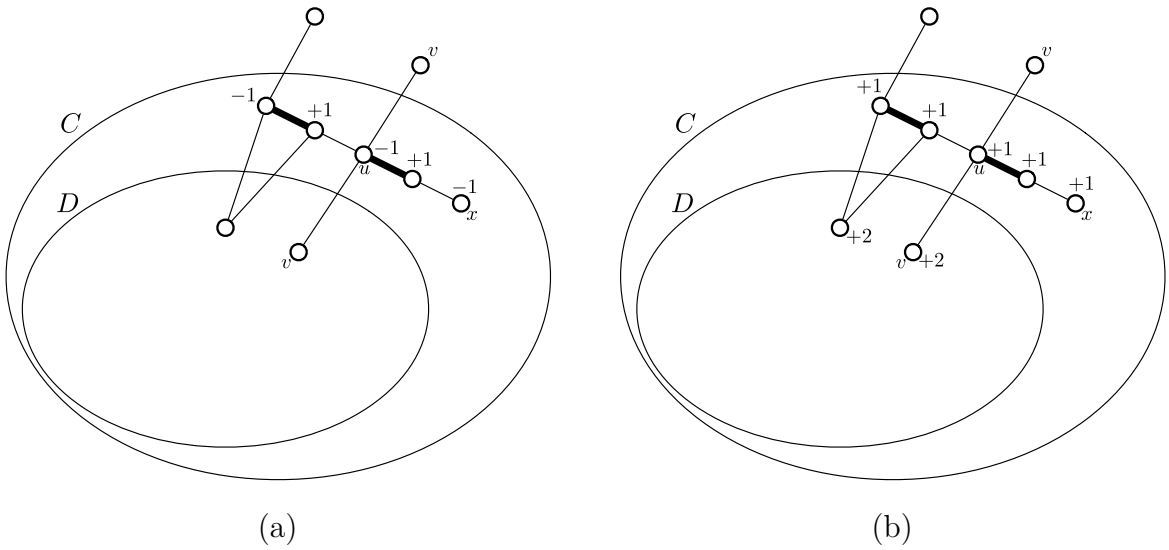


Figure 8: Left: x is a free vertex in the shell (C, D). A dual adjustment increments and decrements some y-values in the shell. If $y(u)$ is decremented this could cause a violation of (near) domination at $yz(u, v)$ when $v \in D$ or $v \notin C$. Right: to restore domination we translate both C and D by 1. This involves decrementing $z(C)$ and $z(D)$ by 2, incrementing $y(v)$ by 2 for all $v \in D$, and incrementing $y(u)$ by 1 for all $u \in C \backslash D$.

Property 8.2.1 and Criterion 1 of eligibility. It is argued that $yz(B)$ is non-increasing over time, and that because $yz(B)$ is within $O(n(B))$ of optimal, after $(n(B))^{\epsilon}$ rounds, at most $O((n(B))^{1-\epsilon})$ free vertices remain in undissolved blossoms inside B. A round that discovers $p \geq 1$ augmenting paths takes $O(m(B) \cdot \min\{\sqrt{n(B)}, p+1\})$ time, so taking $\epsilon = 1/4$ balances the total cost of DismantlePath(B) at $O(m(B)(n(B))^{3/4})$.

The Gabow-Tarjan DismantlePath(B) routine is more efficient but dramatically more complicated in its analysis. They enforce Property 8.2.3 rather than Property 8.2.1 and use Criteria 2 and 3 of eligibility rather than Criterion 1. The upshot is that each round takes near-linear time (rather than up to $O(m(B)\sqrt{n(B)})$ time) but $yz(B)$ is no longer so well behaved! Gabow and Tarjan [61] proved that after all previous calls to DismantlePath(B') complete (where B' is a strict descendant of the major path of B) $yz(B)$ could diverge from the optimum by as much as $O(n(B) \log n(B))$. A blocking flow-type argument shows that after $\sqrt{n(B) \log n(B)}$ rounds there are $O(\sqrt{n(B) \log n(B)})$ free vertices in undissolved old blossoms, leading to a bound of roughly $O(m(B)\sqrt{n(B) \log n(B)})$

for the call to DismantlePath(B). It is straightforward to show that the sum of these costs, over all major path roots $B \in \Omega'$, is roughly $O(m\sqrt{n \log n})$.

8.2.7 A Tour of Our Hybrid mwpm Algorithm

Our algorithm synthesizes ideas from Gabow's original scaling algorithm [54] the simple-minded liquidation approach mentioned in the paper, which yields a running time of $O(m\sqrt{n}\log(nN))$ for all m, n, and N. Our algorithm is based on a few elementary ideas, which must be applied in concert. Blossoms are put in two categories according to a threshold $\tau = \tau(n)$.

Definition 8.2.9. *A blossom B is* small *if $n(B) \leq \tau$ and* large *otherwise, where $\tau = \tau(n)$ is a parameter of the algorithm.*

The number of large root blossoms at any given time is clearly less than n/τ. At the end of scale $i-1$ we guarantee that the sum of large blossoms' z-values is $O(n)$. Thus, liquidating all large blossoms at the beginning of scale i increases $yz(V)$ by $O(n)$, which is tolerable. Small blossoms' z-values are unbounded, but small blossoms have the advantage of being *small*, which allows us to process them using algorithms that would ordinarily be too inefficient.

After all large blossoms are liquidated we reweight the graph, setting

$$\begin{aligned} w(u,v) &\leftarrow w(u,v) - y(u) - y(v) && \text{for all } (u,v) \in E \text{ and} \\ y(u) &\leftarrow 0 && \text{for all } u \in V. \end{aligned}$$

This transformation clearly affects the weight of every perfect matching by the same amount, and therefore does not affect which matchings are mwpms.

We need to cheaply dismantle all small blossoms without screwing up the dual objective $yz(V)$ by more than $O(n)$. By using Gabow's procedure (with Property 8.2.1 and Criterion 1) we can do exactly that in $O(m(B)(n(B))^{3/4})$ time for each maximal small $B \in \Omega'$, which is $O(m\tau^{3/4})$ time in total.

At this point in the algorithm there are no old blossoms, all new blossoms in Ω are necessarily small, and $yz(V)$ is within $O(n)$ of optimal. We now switch from satisfying Property 8.2.1

to Property 8.2.3. It is possible to compute a perfect matching M and blossom set Ω satisfying Property 8.2.3, in $O(m\sqrt{n})$ additional time. However, doing so could violate the assumption that the sum of z-values of large blossoms is $O(n)$. Our solution is to perform τ dual adjustments, which will make the number of free vertices $O(n/\tau)$. Now we switch from satisfying Property 8.2.1 to Property 8.2.3. The first $\sqrt{n} \ll \tau$ dual adjustments are made by calling SEARCHONE(F) $\sqrt{n}$ times on the current set F of all free vertices, using Criterion 2 of eligibility. At this point there are $O(\sqrt{n})$ free vertices. We perform the remaining $\tau - \sqrt{n}$ dual adjustments with calls to BUCKETSEARCH(F), using Criterion 3 of eligibility. Since each call (except possibly the last) finds at least one augmenting path, the total cost to find p augmenting paths is $O(\tau + m(p+1)) = O(m\sqrt{n})$. The standard blocking flow-type argument shows that after τ dual adjustments, there are at most $O(n/\tau)$ free vertices. It is convenient to assume that a scale terminates with a perfect matching, even if it is made perfect artificially. To that end we match each remaining free vertex with an artificial zero weight edge to a new artificial mate.

At the end of the last scale we have a perfect matching, but in a graph where many vertices are matched with artificial edges and mates. In the final *perfection* step we discard all artificial nodes and edges, revealing $O((n/\tau)\log(nN))$ free vertices ($O(n/\tau)$ for each of $\log((n+1)N)$ scales), then run PQSEARCH from each free vertex until an augmenting path is discovered. The total cost per scale is

- $O(n)$ time to liquidate large blossoms and reweight the graph.
- $O(\tau^{3/4}m)$ time to dismantle small blossoms by Gabow's procedure.
- $O(\sqrt{n}m)$ to perform τ dual adjustments.

Together with the time for the perfection step, the total cost of the algorithm is

$$O\left(\left[m\tau^{3/4} + m\sqrt{n} + (n/\tau)(m + n\log n)\right]\log(nN)\right).$$

In order for the cost per scale to be $O(m\sqrt{n})$ we can set τ to be anything in the range $[\sqrt{n}\log n, n^{2/3}]$.

Remark 8.2.10. *One might think the algorithm would be improved (perhaps just in the lower order terms) if we dismantled small blossoms using a variant of the Gabow-Tarjan algorithm using Property 8.2.3 rather than a variant of Gabow's algorithm using Property 8.2.1. Such a substitution would indeed speed up the dismantling of small blossoms, but it could* increase *the dual objective $yz(V)$ by $\Omega(n \log \tau) = \Omega(n \log n)$, which would no longer allow us to reduce the number of free vertices to $O(\sqrt{n})$ in $O(m\sqrt{n})$ time. In this sense Gabow's original algorithm is stronger than Gabow-Tarjan because it guarantees the dual objective is non-increasing over time.*

8.3 The Hybrid Algorithm

The Hybrid algorithm is presented in Figure 9. Recall that τ is the threshold distinguishing large and small blossoms. The graph at the ith scale of Hybrid, G_i, may include many artificial vertices and zero-weight edges accumulated in scales $1, \ldots, i-1$. We use $V = V(G_i)$ and $E = E(G_i)$ to refer to the vertex and edge set of the current graph.

Let w', y', z', M', Ω' be the edge weights, dual variables, matching and blossom set at the end of the previous scale. In the first scale, $w', y', z' = 0$ and $M', \Omega' = \emptyset$, which satisfies Property 8.2.3. Remember that both Ω' and Ω may contain weighted blossoms; the z function counts *all* weighted blossoms in $\Omega \cup \Omega'$ whereas z' only counts Ω'.

8.3.1 Correctness

Lemma 8.3.1. *Consider an edge $e \in V(G_i)$ at scale i.*

- *After Step 2 (Scaling), $w(e) \leq yz(e)$. Moreover, if $e \in M' \cup \bigcup_{B' \in \Omega'} E_{B'}$ then $w(e) \geq yz(e) - 6$. (In the first scale, $w(e) \geq yz(e) - 6$ for every e.)*
- *After Step 4 (Large Blossom Liquidation), $w(e) \leq yz(e) = \sum_{B' \in \Omega' : e \in E(B')} z(B')$. Furthermore, $w(e)$ is even for all $e \in E$ and $y(u)$ is odd for all $u \in V$.*

Therefore, Property 8.2.1 holds after Large Blossom Liquidation.

HYBRID$(G, \hat{w})$

- $G_0 \leftarrow G, y \leftarrow 0, z \leftarrow 0, \Omega \leftarrow \emptyset$.
- For scales $i = 1, \cdots, \lceil \log((n+1)N) \rceil$, run the following steps.

  Initialization

  1. Set $G_i \leftarrow G_{i-1}$, $y' \leftarrow y$, $z' \leftarrow z$, $M \leftarrow \emptyset$, $\Omega' \leftarrow \Omega$, and $\Omega \leftarrow \emptyset$.

  Scaling

  2. Set $w(e) \leftarrow 2(w'(e) + (\text{the } i^{\text{th}} \text{ bit of } \bar{w}(e)))$ for each edge e, set $y(u) \leftarrow 2y'(u) + 3$ for each vertex u, and $z(B') \leftarrow 2z'(B')$ for each $B' \in \Omega'$.

  Large Blossom Liquidation

  3. For each large $B' \in \Omega'$, dissolve B' by setting $y(u) \leftarrow y(u) + z(B')/2$, for each $u \in B'$, then setting $z(B') \leftarrow 0$.
  4. Reweight the graph:
     $$w(u,v) \leftarrow w(u,v) - y(u) - y(v) \qquad \text{for each edge } (u,v) \in E$$
     $$y(u) \leftarrow 0 \qquad \text{for each vertex } u \in V$$

  Small Blossom Dissolution

  5. Run Gabow's Algorithm on each maximal small blossom $B' \in \Omega'$.

  Free Vertex Reduction

  Let F always denote the current set of free vertices and δ the number of dual adjustments performed in Steps 6 and 7.

  6. Run SEARCHONE(F) $\sqrt{n}$ times.
  7. While $\delta < \tau$ and M is not perfect, call BUCKETSEARCH(F), terminating when an augmenting path is found or when $\delta = \tau$.

  Perfection

  8. Delete all artificial free vertices. For each remaining free vertex u, create an artificial $\hat{u}$ with $y(\hat{u}) = \tau$ and a zero-weight matched edge $(u, \hat{u}) \in M$.

- **Finalization** Delete all artificial vertices in $G_{\lceil \log((n+1)N) \rceil}$. For each free vertex u, run PQSEARCH$(\{u\})$ to find an augmenting path matching u.

Figure 9
The HYBRID algorithm.

Proof. At the end of the previous scale, by Property 8.2.3(near domination), $y'z'(e) \geq w'(e) - 2$. After the Scaling step,

$$yz(e) = 2y'z'(e) + 6 \geq 2w'(e) + 2 \geq w(e).$$

If $e \in M' \cup \bigcup_{B' \in \Omega'} E_{B'}$ was an old matching or blossom edge then

$$yz(e) = 2y'z'(e) + 6 \leq 2w'(e) + 6 \leq w(e) + 6.$$

In the first scale, $yz(e) = 6$ and $w(e) = 0$ or 2 for each edge $e \in E$.

Step 3 will increase some yz-values and $w(e) \leq yz(e)$ will be maintained. After Step 4, $w(u, v)$ will decrease by $y(u) + y(v)$, so

$$w(u, v) \leq \sum_{\substack{B' \in \Omega': \\ (u,v) \in E(B')}} z(B').$$

From Property 8.2.3(1) in the previous scale, after Step 2 y-values are odd and z-values are multiples of 4, so y-values remains odd after Step 3. Since initially $w(e)$ is even, $w(e)$ remains even after Step 4. □

Lemma 8.3.2 lists the salient properties of Gabow's algorithm; it is proved in Section 8.3.3.

Lemma 8.3.2. *Suppose that all y-values of the free vertices have the same parity and Property 8.2.1 holds. After calling Gabow's Algorithm on $B \in \Omega'$, where $yz(e) \geq w(e)$ for all $e \in B$ and $yz(e) \leq w(e) + 6$ for all $e \in E_B$, we have:*

- *All the old blossoms $B' \subseteq B$ are dissolved.*
- *The y-values of the free vertices have the same parity and Property 8.2.1 holds.*
- *$yz(V)$ does not increase.*

Futhermore, Gabow's Algorithm runs in $O(m(B)(n(B))^{3/4})$ time.

Lemma 8.3.3. *(Correctness) The* Hybrid *algorithm will return the maximum weight perfect matching of G.*

Proof. First we claim that at the end of each scale i, M is a perfect matching in G_i and Property 8.2.3 is satisfied. By Lemma 8.3.1 and Lemma 8.3.2, Property 8.2.1 is satisfied after the Small Blossom Dissolution step. By Lemma 8.2.7, The Free Vertex Reduction step maintains Property 8.2.3. The perfection step adds/deletes the artificial free vertices and edges to make the matching M perfect. The newly added edges have $w(e) = yz(e)$, and so Property 8.2.3 is maintained at the end of scale i.

Therefore, Property 8.2.3 is satisfied at the end of scale $\lceil \log((n+1)N) \rceil$. By Lemma 8.2.7, each call to PQSearch in the Finalization step also maintains Property 8.2.3 while making the matching perfect. After Finalization, $w(M) \geq w(M^*) - n$. Note that in the last scale $w(e) = 2\bar{w}(e)$ for each edge e, so $\bar{w}(M) \geq \bar{w}(M^*) - n/2$. Since $\bar{w}(e)$ is a multiple of $n+1$, $\bar{w}(M) = \bar{w}(M^*)$. □

8.3.2 Running time

Next, we analyze the running time.

Lemma 8.3.4. *The sum of z-values of large blossoms at the end of a scale is at most $2n$.*

Proof. The Small Blossom Liquidation step only operates on subgraphs of at most τ vertices and therefore cannot create any large blossoms. Every dual adjustment performed in the Free Vertex Reduction step increases the z-values of at most n/τ large root blossoms, each by exactly 2. (The artificial vertices introduced in the Perfection step of previous scales are pendants and cannot be in any blossom.) There are at most τ dual adjustments in Free Vertex Reduction, which implies the lemma. □

Lemma 8.3.5. *Let M' be the matching obtained in the previous scale and M'' be any matching. We have $w(M'') \leq w(M') + 8n - \sum_{u \notin V(M'')} y(u)$ after the Small Blossom Dissolution step of* Hybrid.

Proof. Consider the perfect matching M' obtained in the previous scale, whose blossom set Ω' is partitioned into small and large blossoms, denoted Ω'_{SM} and Ω'_{LG}. For the first scale, we let M' be any perfect matching and $\Omega' = \emptyset$.

When we dissolve a blossom $B' \in \Omega'$, there will be one M'-edge incident to its base such that $yz(u,v)$ increases by $z(B')/2 = z'(B')$. Define K to be the increase in the dual objective due to Large Blossom Liquidation,

$$K = \sum_{B' \in \Omega'_{LG}} z'(B').$$

By Lemma 8.3.4, $K \leq 2n$. Let y_0, z_0 denote the duals right before the Small Blossom Dissolution step. Let y, z, Ω denote the duals and blossom set after Small Blossom Dissolution.

$$\begin{aligned}
w(M') &\geq -6|M'| - K + \sum_{e \in M'} y_0 z_0(e) && \text{Lemma 8.3.1} \\
&= -8n + y_0 z_0(V) && K \leq 2n \\
&\geq -8n + yz(V) && \text{By Lemma 8.3.2} \\
&= -8n + \sum_{u \in V} y(u) + \sum_{B' \in \Omega} z(B') \cdot \lfloor |B'|/2 \rfloor \\
&\geq -8n + \sum_{u \notin V(M'')} y(u) \\
&\qquad + \left(\sum_{u \in V(M'')} y(u) + \sum_{B' \in \Omega} z(B') \cdot \lfloor |B'|/2 \rfloor \right) \\
&\geq -8n + \sum_{u \notin V(M'')} y(u) + \sum_{e \in M''} yz(e) \\
&\geq -8n + \sum_{u \notin V(M'')} y(u) + w(M'') && \text{Property 8.2.1 (domination)}
\end{aligned}$$

□

Lemma 8.3.6. *Let y_5, z_5 be the duals after Step 5, just before the Free Vertex Reduction step. Let M be the matching after Free Vertex Reduction and f be the number of free vertices with respect to M. Suppose that there exists a perfect matching M' such that $w(M) \leq w(M') + 8n - \sum_{u \notin V(M)} y_5(u)$. Then, $f \leq 10n/\tau$.*

Proof. Let y, z, Ω denote the duals and blossom set *after* Free Vertex Reduction. By Prop-

erty 8.2.3 (near domination),

$$
\begin{aligned}
w(M') &\le \sum_{e \in M'} (yz(e) + 2) \\
&= \sum_{u \in V} y(u) + \sum_{e \in M'} \sum_{\substack{B \in \Omega: \\ e \in E(B)}} z(B) + 2|M'| \\
&\le \sum_{u \in V} y(u) + \sum_{B \in \Omega} z(B) \cdot \lfloor |B|/2 \rfloor + 2n && (\#\text{artificial vertices}) \le n \\
&\le \left(\sum_{u \in V(M)} y(u) + \sum_{B \in \Omega} z(B) \cdot \lfloor |B|/2 \rfloor \right) + \sum_{u \notin V(M)} y(u) + 2n \\
&= \sum_{e \in M} yz(e) + \sum_{u \notin V(M)} y(u) + 2n \\
&\le w(M) + \sum_{u \notin V(M)} y(u) + 2n && \text{near tightness} \\
&= w(M) + \sum_{u \notin V(M)} y_5(u) - f\tau + 2n && y(u) = y_5(u) - \tau \\
&= w(M') + 10n - f\tau && \text{by assumption of } M'
\end{aligned}
$$

Therefore, $f\tau \le 10n$, and $f \le 10n/\tau$. □

Therefore, because Lemma 8.3.5 holds for any matching M'', we can apply Lemma 8.3.6 to show the number of free vertices after Free Vertex Reduction is bounded by $O(n/\tau)$.

Theorem 8.3.7. HYBRID *computes an* MWPM *in time*

$$O((m\sqrt{n} + m\tau^{3/4} + (m + n\log n)(n/\tau))\log(nN)).$$

Proof. Initialization, Scaling, and Large Blossom Liquidation still take $O(n)$ time. By Lemma 8.3.2, the Small Blossom Dissolution step takes $O(m(B)(n(B))^{3/4})$ time for each maximal small blossom $B \in \Omega'$, for a total of $O(m\tau^{3/4})$. We now turn to the Free Vertex Reduction step. After $\sqrt{n}$ iterations of SEARCHONE(F), we have performed $\lceil\sqrt{n}\rceil$ units of dual adjustment from all the remaining free vertices. By Lemma 8.3.5 and Lemma 8.3.6, there are at most $10n/\lceil\sqrt{n}\rceil = O(\sqrt{n})$ free vertices. The difference between $w(M)$ and $yz(V)$ is $O(n)$, so we can implement BUCKETSEARCH with an array of $O(n)$ buckets for the priority

queue. A call to BucketSearch(F) that finds $p \geq 0$ augmenting paths takes $O(m(p+1))$ time. Only the last call to BucketSearch may fail to find at least one augmenting path, so the total time for all such calls is $O(m\sqrt{n})$.

By Lemma 8.3.6 again, after the Free Vertex Reduction, there can be at most $10n/\tau$ free vertices. Therefore, in the Finalization step, at most $(10n/\tau)\lceil\log((n+1)N)\rceil$ free vertices emerge after we remove artificial vertices. It takes $O((m + n\log n)(n/\tau)\log(nN))$ time to rematch them with PQSearch [55]. So the total running time is $O((m\sqrt{n} + m\tau^{3/4} + (m + n\log n)(n/\tau))\log(nN))$. □

The running time of Hybrid is not particularly sensitive to the implementation of PQSearch. Setting $\tau = \sqrt{n}\log n$, we get a running time of $O(m\sqrt{n}\log(nN))$ using Gabow's implementation [55] or its slower predecessors [57, 63].

8.3.3 Gabow's Algorithm

The input is a maximal old small blossom $B \in \Omega'$ containing no matched edges, where $yz(e) \geq w(e)$ for all $e \in B$ and $yz(e) \leq w(e)+6$ for all $e \in E_B$. Let T denote the old blossom subtree rooted at B. The goal is to dissolve all the old blossoms in T and satisfy Property 8.2.1 *without increasing the dual objective value* $yz(V)$. Gabow's Algorithm achieves this in $O(m(B)(n(B))^{3/4})$ time. This is formally stated in Lemma 8.3.2.

Gabow's Algorithm decomposes T into major paths. Recall that a child B_1 of B_2 is a *major child* if $|B_1| > |B_2|/2$. A node R is a *major path root* if R is not a major child, so B is a major path root. The *major path* $P(R)$ rooted at R is obtained by starting at R and moving to the major child of the current node, so long as it exists.

Gabow's Algorithm is to traverse each node R in T in *postorder*, and if R is a major path root, to call DismantlePath(R). The outcome of DismantlePath(R) is that all remaining old sub-blossoms of R are dissolved, including R. Define the *rank* of R to be $\lfloor\log n(R)\rfloor$. Suppose that DismantlePath(R) takes $O(m(R)(n(R))^{3/4})$ time. If R and R' are major path roots with the same rank, then they must be disjoint. Summing over all ranks, the total time to

DismantlePath(R): R is a major path root.

Let F be the set of free vertices that are still in undissolved blossoms of R.

1. While $P(R)$ contains undissolved blossoms and $|F| \geq 2$,
   - Sort the undissolved shells in non-increasing order by the number of free vertices, excluding those with less than 2 free vertices. Let $S_1, S_2, \ldots, S_k$ be the resulting list.
   - For $i \leftarrow 1 \ldots k$, call ShellSearch(S_i).
2. If $P(R)$ contains undissolved blossoms (implying $|F| = 1$)
   - Let ω be the free vertex in R. Let $B_1 \subset \cdots \subset B_\ell$ be the undissolved blossoms in $P(R)$ and calculate $T = \sum_i z(B_i)/2$.
   - For $i = 1, 2, \ldots, \ell$, set

     $$y(u) \leftarrow y(u) + z(B_i)/2, \quad \text{for each } u \in B_i,$$
     $$z(B_i) \leftarrow 0.$$

   - Call PQSearch($\{\omega\}$), halting after T dual adjustments.

dissolve B and its sub-blossoms is therefore

$$O\left(\sum_{r=1}^{\lfloor \log n(B) \rfloor} m(B) \cdot (2^{r+1})^{3/4}\right) = O\left((m(B)(n(B))^{3/4}\right).$$

Thus, our focus will be on the analysis of DismantlePath(R).

The procedure DismantlePath(R)

Because DismantlePath is called on the sub-blossoms of B in *postorder*, upon calling DismantlePath(R) the only undissolved blossoms in R are those in $P(R)$. Let $C, D \in P(R) \cup \{\emptyset\}$ with $C \supset D$. The subgraph induced by $C \backslash D$ is called a *shell*, denoted $G(C, D)$. Since all blossoms have an odd number of vertices, $G(C, D)$ is an *even* size shell if $D \neq \emptyset$ and an *odd* size shell if $D = \emptyset$. It is an *undissolved shell* if both C and D are undissolved, or

ShellSearch(C, D)

Let $C^* \supseteq C$ be the smallest undissolved blossom containing C.
Let $D^* \subseteq D$ be the largest undissolved blossom contained in D, or $\emptyset$ if none exists.
Let F^* be the set of free vertices in $G(C^*, D^*)$.

Repeat Augmentation, Blossom Shrinking, Dual Adjustment, and Blossom Dissolution until a halting condition occurs.

- *Augmentation:*
  Extend M to contain an MCM in the subgraph $G(C^*, D^*)$ and update F^*.

- *Blossom Shrinking*:
  Find and shrink blossoms reachable from F^*, exactly as in Edmonds' algorithm.

- *Dual Adjustment*:
  Peform dual adjustments (from F^*) as in Edmonds' algorithm, and perform a unit translation on C^* and D^* as follows:

$$\begin{aligned} z(C^*) &\leftarrow z(C^*) - 2 & \\ z(D^*) &\leftarrow z(D^*) - 2 & \text{if } D^* \neq \emptyset \\ y(u) &\leftarrow y(u) + 2 & \text{for all } u \in D^* \\ y(u) &\leftarrow y(u) + 1 & \text{for all } u \in C^* \setminus D^* \end{aligned}$$

- *Blossom Dissolution*:
  Dissolve root blossoms in Ω with zero z-values as long as they exist. In addition,

  If $z(C^*) = 0$, set $\Omega' \leftarrow \Omega' \backslash \{C^*\}$ and update C^*.

  If $z(D^*) = 0$, set $\Omega' \leftarrow \Omega' \backslash \{D^*\}$ and update D^*.

  Update F^* to be the set of free vertices in $G(C^*, D^*)$.

Halting Conditions:

1. The Augmentation step discovers an augmenting path.
2. $G(C^*, D^*)$ absorbs vertices already searched in the same iteration of DismantlePath.
3. C^* was the outermost undissolved blossom and dissolves during Blossom Dissolution.

Figure 10: ShellSearch(C, D)

C is undissolved and $D = \emptyset$. We call an undissolved shell *atomic* if there is no undissolved blossom $C' \in \Omega'$ with $D \subset C' \subset C$.

The procedure DismantlePath(R) has two stages. The first consists of *iterations*. Each iteration begins by surveying the undissolved blossoms in $P(R)$, say they are $B_k \supset B_{k-1} \supset \cdots \supset B_1$. Let the corresponding atomic shells be $S_i = G(B_i, B_{i-1})$, where $B_0 \stackrel{\text{def}}{=} \emptyset$ and let f_i be the number of free vertices in S_i. We sort the (S_i) in non-increasing order by their number of free vertices and call ShellSearch(S_i) in the order, but refraining from making the call unless S_i contains at least two free vertices.

The procedure ShellSearch(C, D) is simply an instantiation of EdmondsSearch with the following features and differences.

1. There is a *current atomic shell* $G(C^*, D^*)$, which is initially $G(C, D)$, and the Augmentation, Blossom Formation, and Dual Adjustment steps only search from the set of free vertices in the current atomic shell. By definition C^* is the smallest undissolved blossom containing C and D^* the largest undissolved blossom contained in D, of $\emptyset$ if no such blossom exists.

2. An edge is eligible if it is tight (Criterion 1) and in the current atomic shell. (Tight edges that straddle the shell are specifically excluded.)

3. Each unit of dual adjustment is accompanied by a unit translation of C^* and D^*, if $D^* \neq \emptyset$. This may cause either/both of C^* and D^* to dissolve if their z-values become zero, which then causes the current atomic shell to be updated.

4. Like EdmondsSearch, ShellSearch halts after the first Augmentation step that discovers an augmenting path. However, it halts in two other situations as well. If C^* is the outermost undissolved blossom in $P(R)$ and C^* dissolves, ShellSearch halts immediately. If the current shell $G(C^*, D^*)$ ever intersects a shell searched in the same iteration of DismantlePath(R), ShellSearch halts immediately. Therefore, at the end of an iteration of DismantlePath(R), every undissolved atomic shell contains at least two vertices that were matched (via an augmenting path) in the iteration.

Blossom translations are used to preserve Property 8.2.1(domination) for all edges, specifically those crossing the shell boundaries. We implement ShellSearch(C, D) using an

array of buckets for the priority queue, as in BucketSearch, and execute the Augmentation step using the Micali-Vazirani [122, 169] algorithm or Gabow-Tarjan cardinality matching algorithm [61, §10]. Let t be the number of dual adjustments, $G(C^*, D^*)$ be the current atomic shell before the last dual adjustment, and $p \geq 0$ be the number of augmenting paths discovered before halting. The running time of ShellSearch(C, D) is $O(t+m(C^*, D^*) \cdot \min\{p+1, \sqrt{n(C^*, D^*)}\})$. We will show that t is bounded by $O(n(C^*, D^*)))$ as long as the number of free vertices inside $G(C^*, D^*)$ is at least 2. See Corollary 8.3.13.

The first stage of DismantlePath(R) ends when either all old blossoms in $P(R)$ have dissolved (in which case it halts immediately) or there is exactly one free vertex remaining in an undissolved blossom. In the latter case we proceed to the second stage of DismantlePath(R) and liquidate all remaining old blossoms. This preserves Property 8.2.1 but screws up the dual objective $yz(R)$, which must be corrected before we can halt. Let ω be the last free vertex in an undissolved blossom in R and $T = \sum_i z(B_i)/2$ be the aggregate amount of translations performed when liquidating the blossoms. We perform PQSearch$(\{\omega\})$, halting after exactly T dual adjustments. The search is guaranteed not to find an augmenting path. It runs in $O(m(R) + n(R) \log n(R))$ time [55].

To summarize, DismantlePath(R) dissolves all old blossoms in $P(R)$, either in stage 1, through gradual translations, or in stage 2 through liquidation. Moreover, Property 1 is maintained throughout DismantlePath(R). In the following, we will show that DismantlePath(R) takes $O(m(R)(n(R))^{3/4})$ time and the dual objective value $yz(S)$ does not increase for every S such that $R \subseteq S$. In addition, we will show that at all times, the y-values of all free vertices have the same parity.

Properties

We show the following lemmas to complete the proof of Lemma 8.3.2. Let y_0, z_0 denote the initial duals, before calling Gabow's Algorithm.

Lemma 8.3.8. *Throughout* DismantlePath(R)*, we have $y(u) \geq y_0(u)$ for all $u \in R$. Moreover, the y-values of free vertices in R are always odd.*

Proof. We will assume inductively that this holds after every recursive call of DismantlePath(R') for every R' that is a non-major child of a $P(R)$ node. Then, it

suffices to show $y(u)$ does not decrease and the parity of free vertices always stays the same during DismantlePath(R). Consider doing a unit of dual adjustment inside the shell $G(C^*, D^*)$. Every vertex in D^* has its y-value increased by 2, every vertex in C^* either has its y-value unchanged or increased by 1 or 2. The y-values of the free vertices in C^* remain unchanged.

Consider that in the second stage of DismantlePath(R), when artificially liquidating blossom B_i, $y(\omega)$ increases by $z(B_i)/2$. Therefore, $y(\omega)$ increases by T before the call to PQSearch($\{\omega\}$). Define $w'(u, v) = yz(u, v) - w(u, v)$. The eligible edges must have $w'(u, v) = 0$. We can easily see that when we dissolve B_i and increase the y-values of vertices in B_i, the w'-distance from ω to any vertex outside the largest undissolved blossom B_ℓ increases by $z(B_i)/2$. Therefore, the total distance from ω to any vertex outside B_ℓ increases by T after dissolving all the blossoms. Since every other vertex inside B_ℓ is matched, PQSearch($\{\omega\}$) will perform T dual adjustments and halt before finding an augmenting path. We conclude that $y(\omega)$ is restored to the value it had before the second stage of DismantlePath(R). □

Lemma 8.3.9. *If Property 8.2.1 (granularity, tightness, and domination) and Property 8.2.8 are satisfied and y-values of the free vertices have the same parity, then Property 8.2.1(granularity, tightness, and domination) and Property 8.2.8 hold after calling* ShellSearch(C, D).

Proof. First we will argue that Property 8.2.1 holds after calling ShellSearch(C, D). The current atomic shell $G(C^*, D^*)$ cannot contain any old blossoms, since we are calling DismantlePath(R) in postorder. Because we are simulating EdmondsSearch(F^*) from the set F^* of free vertices in $G(C^*, D^*)$, whose y-values have the same parity, by Lemma 8.2.7, Property 8.2.1 holds in $G(C^*, D^*)$. It is easy to check that Property 8.2.1(1,2) (granularity, active blossoms) hold in G. Now we only need to check Propery 8.2.1(3,4) (domination and tightness) for the edges crossing C^* or D^*. By Property 8.2.8, there are no crossing matched edges and all the newly created blossoms lie entirely in $G(C^*, D^*)$. Therefore, tightness must be satisfied. The translations on blossoms C^* and D^* keep the yz-values of edges straddling $C^* \backslash D^*$ non-decreasing, thereby preserving domination.

Now we claim Property 8.2.8 holds. We only consider the effect on the creation of new blossoms, since the dissolution of C^* or D^* cannot violate Property 8.2.8. Since edges

straddling the atomic shell $G(C^*, D^*)$ are automatically ineligible, we will only create new blossoms inside $G(C^*, D^*)$. Since $G(C^*, D^*)$ does not contain any old blossoms and the new blossoms in $G(C^*, D^*)$ form a laminar set, Property 8.2.8(1,2) hold. Similarly, the augmentation only takes place in $G(C^*, D^*)$ which does not contain old blossoms, Property 8.2.8(3) holds. □

Lemma 8.3.10. *Consider an execution of* EdmondsSearch(F) *when Property 8.2.1 is satisfied, using Criterion 1 of eligibility. Each unit of dual adjustment decreases $yz(S)$ by $|F|$, if $S \supseteq \hat{V}_{in} \cup \hat{V}_{out}$.*

Proof. Since $\hat{V}_{in} \cup \hat{V}_{out} \subseteq S$, no matched edges cross S. By Property 8.2.1(tightness) $yz(S) = w(M \cap S) + \sum_{u \in S \setminus V(M)} y(u)$. A dual adjustment preserves tightness, reducing the second term by $|F|$. □

Lemma 8.3.11. *For any S such that $R \subseteq S$, $yz(S)$ never increases throughout* DismantlePath(R).

Proof. Consider a dual adjustment in ShellSearch(C, D). Let F^* be the set of free vertices in the current atomic shell $G(C^*, D^*)$. By Lemma 8.3.10, the search inside $G(C^*, D^*)$ decreases $yz(S)$ by $|F^*|$. The translation on C^* increases $yz(S)$ by 1. If $D^* \neq \emptyset$, the translation of D^* also increases $yz(S)$ by 1. Therefore, a dual adjustment in ShellSearch decreases $yz(S)$ by $|F^*| - 2$, if $D^* \neq \emptyset$, and by $|F^*| - 1$ if $D = \emptyset$. Since $G(C^*, D^*)$ contains at least 2 free vertices, $yz(S)$ does not increase during DismantlePath(R).

Suppose the second stage of DismantlePath(R) is reached, that is, there is exactly one free vertex ω in an undissolved blossom in R. When we liquidate all remaining blossoms in R, $yz(S)$ increases by T. As shown in the proof of Lemma 8.3.8, PQSearch$(\{\omega\})$ cannot stop until it reduces $yz(\omega)$ by T. By Lemma 8.3.10 this decreases $yz(S)$ by T, thereby restoring $yz(S)$ back to its value before the second stage of DismantlePath(R). □

The following lemma considers a *not necessarily atomic* undissolved shell $G(C, D)$ at some point in time, which may, after blossoms dissolutions, become an atomic shell. Specifically, C and D are undissolved but there could be *many* undissolved $C' \in \Omega'$ for which $D \subset C' \subset C$.

Lemma 8.3.12. *Consider a call to* DismantlePath(R) *and any shell* $G(C, D)$ *in* R. *Throughout the call to* DismantlePath, *so long as* C *and* D *are undissolved (or* C *is undissolved and* $D = \emptyset$*)* $yz(C) - yz(D) \geq y_0z_0(C) - y_0z_0(D) - 3n(C \setminus D)$.

Proof. If $D = \emptyset$, we let D' be the singleton set consisting of an arbitrary vertex in C. Otherwise, we let $D' = D$. Let ω be a vertex in D'. Due to the structure of the blossoms, we can find a perfect matching M_ω that is also perfect when restricted to $D' \setminus \{\omega\}$ or $C' \setminus D'$, for any $C' \in \Omega'$ with $C' \supset D'$. By Lemma 8.3.1, every $e \in M_\omega \cap E_R$ has $y_0z_0(e) \leq w(e) + 6$. Therefore,

$$\begin{aligned}\sum_{e \in M_\omega \cap E(C \setminus D')} w(e) &\geq \sum_{e \in M_\omega \cap E(C \setminus D')} y_0z_0(e) - 6n(C \setminus D')/2 \\ &= \sum_{u \in V(C \setminus D')} y_0(u) + \sum_{\substack{C' \in \Omega': \\ D' \subset C'}} z_0(C') \cdot \frac{|C' \cap C| - |D'|}{2} - 3n(C \setminus D') \\ &= y_0z_0(C) - y_0z_0(D') - 3n(C \setminus D').\end{aligned}$$

On the other hand, by the domination condition of Property 8.2.1, we have

$$\begin{aligned}&\sum_{e \in M_\omega \cap E(C \setminus D')} w(e) \\ &\leq \sum_{e \in M_\omega \cap E(C \setminus D')} yz(e) \\ &= \sum_{u \in V(C \setminus D')} y(u) + \sum_{\substack{C' \in \Omega': \\ D' \subset C'}} z(C') \cdot \frac{|C' \cap C| - |D'|}{2} + \sum_{B \in \Omega} z(B) \cdot |M_\omega \cap E(B \cap C \setminus D')| \\ &\leq \sum_{u \in V(C \setminus D')} y(u) + \sum_{\substack{C' \in \Omega': \\ D' \subset C'}} z(C') \cdot \frac{|C' \cap C| - |D'|}{2} + \sum_{\substack{B \in \Omega: \\ B \subset C}} z(B) \cdot \lfloor \frac{|B| - |B \cap D'|}{2} \rfloor\end{aligned}$$

Consider a $B \in \Omega$ that contributes a non-zero term to the last sum. By Property 8.2.8, $\Omega \cup \Omega'$ is laminar so either $B \subseteq D$ or $B \subseteq C \setminus D$. In the first case B contributes nothing to the sum. In the second case we have $|B \cap D'| \leq 1$ (it can only be 1 when $D = \emptyset$ and D' is a singleton set intersecting B) so it contributes exactly $z(B) \cdot \lfloor |B|/2 \rfloor$. Continuing on,

$$
\begin{aligned}
&= \sum_{u \in V(C \setminus D')} y(u) + \sum_{\substack{C' \in \Omega': \\ D' \subset C'}} z(C') \cdot \frac{|C' \cap C| - |D'|}{2} + \sum_{\substack{B \in \Omega: \\ B \subset (C \setminus D)}} z(B) \cdot \lfloor \frac{|B|}{2} \rfloor \\
&= yz(C) - yz(D').
\end{aligned}
$$

Therefore, $yz(C) - yz(D') \geq y_0 z_0(C) - y_0 z_0(D') - 3n(C, D')$. When $D = \emptyset$ we have $yz(D') = y(\omega) \geq y_0(\omega)$. Therefore, regardless of D, $yz(C) - yz(D) \geq y_0 z_0(C) - y_0 z_0(D) - 3n(C, D)$. □

Corollary 8.3.13. *The number of dual adjustment in* ShellSearch(C, D) *is bounded by* $O(n(C^* \setminus D^*))$ *where* $G(C^*, D^*)$ *is the current atomic shell when the last dual adjustment is performed.*

Proof. We first claim that the recursive call of DismantlePath(R') on the descendants R' of $P(R)$ does not decrease $yz(C^*) - yz(D^*)$. If $R' \subset D^*$, then any dual adjustments done in DismantlePath(R') changes $yz(C^*)$ and $yz(D^*)$ by the same amount. Otherwise, $R' \subset G(C^*, D^*)$. In this case, DismantlePath(R') has no effect on $yz(D^*)$ and does not increase $yz(C^*)$ by Lemma 8.3.10. Therefore, $yz(C^*) - yz(D^*) \leq y_0 z_0(C^*) - y_0 z_0(D^*)$.

First consider the period in the execution of ShellSearch(C, D) when $D^* \neq \emptyset$. During this period ShellSearch performs some number of dual adjustments, say k. There must exist at least two free vertices in $G(C^*, D^*)$ that participate in all k dual adjustments. Note that a unit translation on an old blossom $C'' \in \Omega'$, where $D^* \subseteq C'' \subseteq C^*$, has no net effect on $yz(C^*) - yz(D^*)$, since it increases both $yz(C^*)$ and $yz(D^*)$ by 1. By Lemma 8.3.10, $yz(C^*) - yz(D^*)$ decreases by at least $2k$ due to the dual adjustments. By Lemma 8.3.12, $yz(C^*) - yz(D^*)$ decreases by at most $3n(C^* \setminus D^*)$, and so $k \leq 3/2 \cdot n(C^* \setminus D^*)$.

Now consider the period when $D^* = \emptyset$. Let $G(C', D')$ to be the current atomic shell just before the smallest undissolved blossom D' dissolves and let k' be the number of dual adjustments performed in this period, after D' dissolves. By Lemma 8.3.10, all prior dual adjustments have not increased $yz(C^*)$. There exists at least 3 free vertices in C^* that participate in all k' dual adjustments. Each translation of C^* increases $yz(C^*)$ by 1. By

Lemma 8.3.10, $yz(C^*)$ decreases by at least $3k' - k' = 2k'$ due to the k' dual adjustments and translations performed in tandem. By Lemma 8.3.12, $yz(C^*)$ can decrease by at most $3n(C^*)$, so $k' \leq 3/2 \cdot n(C^*)$. The total number of dual adjustments is therefore $k + k' \leq 3/2(n(C' \setminus D') + n(C^*)) < 3n(C^*)$. □

The following two lemmas are adapted from [54, Lemmas 2.4 and 2.5].

Lemma 8.3.14. *For any $\epsilon > 0$, the number of iterations of* DismantlePath(R) *with $|F| \geq (n(R))^\epsilon$ is $O((n(R))^{1-\epsilon})$.*

Proof. Consider an iteration in DismantlePath(R). Let f be the number of free vertices before this iteration. Call an atomic shell *big* if it contains more than 2 free vertices. We consider two cases depending on whether more than $f/2$ vertices are in the big atomic shells or not. Suppose big shells do contain more than $f/2$ vertices. The free vertices in an atomic shell will not participate in any dual adjustment only if some adjacent shells have dissolved into it. When a shell containing f' free vertices dissolves into (at most 2) adjacent shells and the call to ShellSearch immediately halts, it must be the case that it prevents at most $2f'$ other free vertices from participating in a dual adjustment, due to the order we search the shells. Therefore, at least $f/6$ free vertices in the big shells participate in at least one dual adjustment. Let S_i be a big even shell with f_i free vertices. If they are subject to a dual adjustment then $yz(R)$ decreases by at least $(f_i - 2) \geq f_i/2$ by Lemma 8.3.10, since the shell is big. If S_i is a big *odd* shell then the situation is even better: by Lemma 8.3.10 $yz(R)$ is reduced by $(f_i - 1) \geq \frac{2}{3} f_i$. Therefore, $yz(R)$ decreases by at least $f/12$.

The case when more than $f/2$ free vertices are in small atomic shells can only happen $O(\log n)$ times. In this case, there are at least $\lfloor f/4 \rfloor$ small shells. In each shell, there must be vertices that were matched during the previous iteration. Therefore, in the previous iteration, there must have been at least $f + 2\lfloor f/4 \rfloor$ free vertices. This can only happen $O(\log n)$ times, since the number of free vertices shrinks by a constant factor each time it happens.

By Lemma 8.3.11, $yz(R)$ does not increase in the calls to DismantlePath on the descendants of $P(R)$. By Lemma 8.3.12, since $yz(R)$ decreases by at most $3n(R)$, the number of iterations with $|F| \geq (n(R))^\epsilon$ is at most $12(n(R))^{1-\epsilon} + O(\log n) = O((n(R))^{1-\epsilon})$. □

Lemma 8.3.15. DismantlePath(R) *takes at most $O((m(R)n(R))^{3/4})$ time.*

Proof. Recall that ShellSearch is implemented just like BucketSearch, using an array for a priority queue. This allows all operations (insert, deletemin, decreasekey) to be implemented in $O(1)$ time, but incurs an overhead *linear* in the number of dual adjustments/buckets scanned. By Corollary 8.3.13 this is $\sum_i O(n(S_i)) = O(n(R))$ per iteration. By Lemma 8.3.14, there are at most $O((n(R))^{1/4})$ iterations with $|F| \geq (n(R))^{3/4}$. Consider one of these iterations. Let $\{S_i\}$ be the shells in the end of iteration. The augmentation takes $\sum_i O(m(S_i)\sqrt{n(S_i)}) = O(m(R)\sqrt{n(R)})$ time. Therefore, the total time of these iterations is $O(m(R)(n(R))^{3/4})$. There can be at most $(n(R))^{3/4}$ more iterations afterwards, since each iteration matches at least 2 free vertices. Therefore, the cost for all subsequent Augmentation steps is $O(m(R)(n(R))^{3/4})$. Finally, the second stage of DismantlePath(R), when one free vertex in an undissolved blossom remains, takes $O(m(R) + n(R)\log n(R))$ time [55]. Therefore, the total running time of DismantlePath(R) is $O(m(R)(n(R))^{3/4})$. □

Let us summarize what has been proved. By the inductive hypothesis, all calls to DismantlePath preceding DismantlePath(R) have (i) dissolved all old blossoms in R excluding those in $P(R)$, (ii) kept the y-values of all free vertices in R the same parity (odd) and kept $yz(R)$ non-increasing, and (iii) maintained Properties 8.2.1 and 8.2.8. If these preconditions are met, the call to DismantlePath(R) dissolves all remaining old blossoms in $P(R)$ while satisfying (ii) and (iii). Futhermore, DismantlePath(R) runs in $O(m(R)(n(R))^{3/4})$ time. This concludes the proof of Lemma 8.3.2.

8.4 Conclusion

We have presented a new scaling algorithm for MWPM on general graphs that runs in $O(m\sqrt{n}\log(nN))$ time, which is the first significant improvement to Gabow and Tarjan's 1991 algorithm [61]. Our algorithm is *momentarily optimal* in the sense that, before it can be improved, one would have to first improve the *bipartite* weighted matching algorithms [38, 60, 68, 138], which also run in $O(m\sqrt{n}\log(nN))$ time, but are much simpler due to the absence of blossoms. Moreover, each scale of our algorithm runs in $O(m\sqrt{n})$ time, matching the best MCM algorithm for general graphs [122, 169], so up to the $O(\log(nN))$ factor, ours cannot be improved without first improving the Micali-Vazirani algorithm. These observations are merely meant to point out algorithmic barriers, not claim hard lower bounds.

Indeed, Mądry's [123] recent $\tilde{O}(m^{10/7})$-time algorithm for max-flow on unit-capacity networks implies that MCM on bipartite graphs can be solved in the same time bound. The logical next step is to generalize it to min-cost flow (and hence bipartite MWPM), and eventually to MWPM on general graphs, which is a bidirected flow problem.

We are not aware of any experimental implementations of either Gabow's algorithm [54] or the Gabow-Tarjan algorithm [61], so it remains an open question whether the scaling technique can offer any *practical* speedups over implementations of Edmonds' algorithm.

Chapter 9

A Scaling Algorithm for Maximum Weight Matching in Bipartite Graphs

9.1 Introduction

The input is a weighted bipartite graph $G = (V, E, w)$ where $|V| = 2n$, $|E| = m$, $w : E \to \mathbb{R}$. A *matching* M is a set of vertex-disjoint edges. The *maximum weight matching (*MWM*)* problem is to find a matching M such that $w(M) = \sum_{e \in M} w(e)$ is maximized among all matchings, whereas the *maximum weight perfect matching (*MWPM*)* problem requires every vertex to be matched.

The MWPM problem and the MWM are reducible to each other [60]. To reduce from the problem of MWM to MWPM, obtain $\tilde{G}$ by making two copies of G and add a zero weight edge between each two copies of vertex. Then, $\tilde{G}$ is still bipartite and a MWPM in $\tilde{G}$ gives a MWM in G. Conversely, to reduce from the problem of MWPM to MWM, we simply add nN to the weight of each edge, where N is the maximum weight of the edges. This will guarantee that the MWM found is perfect.

Figure 1 shows the previous results on these problems. The first procedure for solving the MWPM problem dates back to 150 years ago by Jacobi [85]. However, the procedure was not discovered until recently [137]. In the 1950s, Kuhn [112] and Munkres [131] developed the "Hungarian" algorithm to solve the MWPM problem, where the former credited it to the earlier works of König and Egerváry. Later in [6], Balinski and Gomory gave an alternate approach to this problem, the primal method. The previous approaches grow the matching

from empty while maintaining the feasibility of the dual program. In contrast, the primal method maintains the perfect matching from the beginning and fixes the infeasible dual solution along the way.

Bipartite Weighted Matching

Year	Author	Problem	Running Time	Notes
1865 1955 1957 1964	Jacobi Kuhn Munkres Balinski & Gomory	MWPM	$\text{poly}(n)$	
1970 1971	Edmonds & Karp$^{(\star)}$ Tomizawa$^{(\star)}$	MWPM	$mn \log n$	Using binary heaps
1977	Johnson$^{(\star)}$	MWPM	$mn \log_d n$	Using d-ary heaps, $d = 2m/n$
1983	Gabow	MWPM MWM	$mn^{3/4} \log N$ $Nm\sqrt{n}$	$N =$ max. *integer* weight
1984	Fredman & Tarjan$^{(\star)}$	MWPM	$mn + n^2 \log n$	Using Fibonacci heaps
1988 1992	Gabow & Tarjan Orlin & Ahuja	MWPM	$m\sqrt{n} \log(nN)$	*integer* weights
1996	Cheriyan & Mehlhorn	MWPM	$n^{2.5} \log(nN)(\frac{\log \log n}{\log n})^{1/4}$	
1999	Kao, Lam, Sung & Ting	MWM	$Nm\sqrt{n}(\frac{\log(n^2/m)}{\log n})$	*integer* weights
2002	Thorup$^{(\star)}$	MWPM	mn	*integer* weights, randomized
2003	Thorup$^{(\star)}$	MWPM	$mn + n^2 \log \log n$	*integer* weights
2006	Sankowski	MWPM	Nn^{ω}	*integer* weights, randomized $\omega =$ matrix mult. exponent
new result		MWM	$m\sqrt{n} \log N$	*integer weights*

Table 4: Previous results on the MWPM and MWM problems. Algorithms that solve MWPM also solve MWM with the same running time. Conversely, algorithms that solve MWM can be used to solve MWPM, while the factor N becomes nN in the running time. (*) denotes implementations of the Hungarian algorithm using different priority queues.

Later, Edmonds and Karp [44] and, independently, Tomizawa [167], observed that implementing the Hungarian algorithm for MWPM amounted to computing single-source shortest paths n times on a nonnegatively weighted graph. The running time of their algorithm depends on the best implementation of Dijkstra's algorithm, which has been improved over time [50, 91, 163, 164].

Faster algorithms are known when the edge weights are bounded integers in $[-N, N]$, and a word RAM model with $\log n + \log N$ word size is assumed. Gabow [53] gave a scaling

approach for the MWPM problem, where he also showed the MWM problem can be solved in $O(Nm\sqrt{n})$ time. Gabow and Tarjan [60] improved the scaling approach to solve the MWPM in $O(m\sqrt{n}\log(nN))$ time. Later, Orlin and Ahuja [138] gave another algorithm with the same running time.

There are several faster algorithms for dense graphs. Cheriyan and Mehlhorn [20] exploited the RAM model and used a bit compression technique to implement Orlin and Ahuja's algorithm. Kao et al. [93] showed that the MWM problem can be decomposed into MWM problems with uniform weights, where a faster algorithm for the maximum cardinality matching problem in [49] can be applied. Extending from [79, 130], Sankowski gave an algebraic approach to solve this problem [155]. For general graphs, the relevant works are in [42, 52, 54, 55, 57, 61, 63, 79, 130]

In this paper, we look at the MWM problem with bounded integers in $[0, N]$, because negative weights can always be ignored. We present a new scaling algorithm that runs in $O(m\sqrt{n}\log N)$ time. Our algorithm improves the previous bound of $O(Nm\sqrt{n})$ by Gabow [53] and $O(m\sqrt{n}\log(nN))$ by Gabow and Tarjan [60], which stood for over 20 years. Other algorithms by [93] and [155] are not strongly polynomial and outperform ours only when $N = O(1)$ and the graph is very dense. The former requires $m = \omega(n^{2-\epsilon})$ for any $\epsilon > 0$, whereas the latter requires $m = \omega(n^{\omega-1/2})$, which is $\omega(n^{1.876})$ by the current fastest matrix multiplication technology [25]. Though our improvement is small, it indicates that the MWM problem might be easier than the MWPM problem.

Our approach consists of three phases. The first phase uses a search similar to one iteration of [60] to find a good initial matching. The second phase is the scaling phase. In contrast to [60], where they run up to $\log(nN)$ scales to ensure the solution is optimal, we run only up to $\log N$ scales. Then, the third phase makes the solution optimal by fixing the absolute error left by the first two phases. In some sense, our first and third phase have the effect equivalent to $0.5\log n$ scales of the Gabow-Tarjan algorithm [60], thereby saving the additional $\log n$ scales. Like Balinski and Gomory's algorithm [6], our algorithm adjusts the matching throughout the second and third phases instead of finding a new one in each scale. In addition, as in [66], we bound our running time by using Dilworth's Lemma, in particular, that every partial order has a chain or anti-chain of size $\Omega(\sqrt{n})$.

9.1.1 Definitions and Preliminaries

A *matching* M is a set of vertex-disjoint edges. A vertex is *free* if it is not incident to an M edge, otherwise it is *matched*. If a vertex u is matched, denote its mate by u'. The MWM problem can be expressed as the following integer linear program, where x represents the incidence vector of a matching.

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} w(e)x(e) & & \\ \text{subject to} \quad & 0 \le x(e) \le 1,\ x(e) \text{ is an integer} & \forall e \in E & \qquad (9.1) \\ & \sum_{e=uv \in E} x(e) \le 1 & \forall u \in V & \end{aligned}$$

It was shown that basic solutions to the linear program are integral. The dual of the linear program is as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V} y(u) & & \\ \text{subject to} \quad & y(e) \ge w(e) & \forall e \in E & \qquad (9.2) \\ & y(u) \ge 0 & \forall u \in V & \\ \text{where we define} \quad & y(uv) \stackrel{\text{def}}{=} y(u) + y(v) & & \end{aligned}$$

By the complementary slackness condition, M and y are optimal iff $\forall e \in M$, $y(e) = w(e)$ and for all free vertices u, $y(u) = 0$. In the MWPM problem, the third inequality in the LP becomes equality, $\sum_{e=uv \in E} x(e) = 1, \forall u \in V$. Therefore, the condition $y(u) \ge 0, \forall u \in V$ is dropped in the dual program. If $\forall e \in M$, $y(e) = w(e)$, then M and y are optimal.

Definition 9.1.1. *Given δ_0, let $\delta_i = \delta_0/2^i$, $w_i(e) = \delta_i \lfloor w(e)/\delta_i \rfloor$. The eligibility graph $G[c,d]$ at scale i is the subgraph of G containing all edges e satisfying either $e \notin M$ and $y(e) = w_i(e)$ or $e \in M$ and $w_i(e) + c\delta_i \le y(e) \le w_i(e) + d\delta_i$.*

An alternating path (or cycle) is one whose edges alternate between M and $E \setminus M$. Our algorithm consists of three phases, and we let $\delta_0 = 2^{\lfloor \log(N/\sqrt{n}) \rfloor}$ and the number of scales $L = \lceil \log N \rceil$, so that $w(e) = w_L(e)$ for all $e \in E$. An *augmenting walk/path/cycle* is defined differently in each phase:

1. Phase I: The phase operates at scale 0. An augmenting walk refers to an alternating path in $G[1,1]$ with free endpoints. For convenience, call such a path an augmenting path.

2. Phase II: The phase operates at scales $1 \dots L$. An augmenting walk is either an alternating cycle in $G[1,3]$ or an alternating path in $G[1,3]$ whose end vertices have 0 y-values. For convenience, call the former an augmenting cycle and the latter an augmenting path. Notice that an endpoint of an augmenting path can be either free or matched. If an endpoint is matched, then we require its mate to be contained in the path as well.

3. Phase III: The phase operates at scale L. An augmenting walk is in $G[0,1]$ and defined the same as Phase II with one more restriction: The walk P must contain at least one matched edge that is not tight. That is, $y(e) \neq w_L(e)$, for some $e \in P \cap M$.

Given an augmenting walk P, by augmenting M along P, we get a matching $M \oplus P = (M \setminus P) \cup (P \setminus M)$. Given a subgraph $H \subseteq G$ and a vertex set $X \subseteq V$, let $V_{odd}(X, H)$ denote the set of vertices reachable through an odd-length alternating path in H starting with an **unmatched edge** that incidents to a vertex in X, and $V_{even}(X, H)$ be the set reachable via an even-length alternating path. For convenience, sometimes we denote a singleton $\{x\}$ by x. Let $\overrightarrow{G}$ denote the directed graph obtained by orienting edges e from left to right if $e \notin M$, from right to left if $e \in M$. Every alternating path in G must be a path in $\overrightarrow{G}$ and vice versa.

9.2 Algorithm

Property 9.2.1. *Throughout scale $i \in [0, L]$, we maintain matching M and dual variables y satisfying the following:*

1. ***(Granularity of y)*** *$y(u)$ is a nonnegative multiple of δ_i.*

2. ***(Domination)*** *$y(e) \geq w_i(e)$ for all $e \in E$.*

3. ***(Near Tightness)*** *$y(e) \leq w_i(e) + 3\delta_i$ for $e \in M$. At the end of scale i, it is tightened so that $y(e) \leq w_i(e) + \delta_i$ for $e \in M$.*

4. ***(Free Vertex Duals)*** *The y-values of free vertices are 0 at the end of scale i.*

Lemma 9.2.2. *Let M^* be the optimal matching. If M and y satisfy Property 9.2.1 at the end of the scale L, then $w(M) \geq w(M^*) - n\delta_L$. Furthermore, when M is perfect and M^* is the optimal perfect matching, the same inequality holds if $y(e) \geq w(e)$ for all $e \in E$ and $y(e) \leq w(e) + \delta_L$ for $e \in M$.*

Proof.

$$\begin{aligned}
w(M) &= \sum_{e \in M} w(e) \\
&\geq \sum_{e \in M} y(e) - n\delta_L && \text{near tightness} \\
&= \sum_{u \in V} y(u) - n\delta_L && \text{free vertex duals} \\
&\geq \sum_{e \in M^*} y(e) - n\delta_L && \text{non-negativity} \\
&\geq \sum_{e \in M^*} w(e) - n\delta_L && \text{domination} \\
&= w(M^*) - n\delta_L
\end{aligned}$$

If M and M^* are perfect, then we can skip from the second line to the fourth line, since $\sum_{e \in M} y(e) - n\delta_L = \sum_{e \in M^*} y(e) - n\delta_L$. □

The goal of each phase is as follows. Phase I finds the initial matching and dual variables satisfying Property 9.2.1 for scale $i = 0$. Phase II maintains Property 9.2.1 after entering from scale $i - 1$ to i, for $i \in [1, L]$. In particular, we want to have $y(e) \leq w_i(e) + \delta_i$ for all $e \in M$ at the end of each scale. Phase III tightens the near tightness condition to exact tightness for all $e \in M$ after scale L so that $y(e) = w_L(e) = w(e)$ for all $e \in M$.

9.2.1 Phase I

In this phase, our algorithm will be working on $G[1,1]$ so that if one augments along an augmenting walk, all edges of the walk become ineligible.

Our algorithm maintains an invariant: All free left vertices, F, have equal and minimal y-values among left vertices and all free right vertices have zero y-values. After the initialization, Property 9.2.1(4) is violated. We fix it by repeating the augmentation/dual adjustment steps until all vertices in F have zero y-values. The procedure described in the pseudocode is a modified Gabow-Tarjan algorithm [60] for one scale, where we always adjust dual variables by δ_0 in each iteration and stop when free vertices have zero y-values rather than when the matching is perfect.

Initialization:
$M \leftarrow \emptyset$.
Set $y(v) \leftarrow \begin{cases} 2^{\lfloor \log N \rfloor} & \text{if } v \text{ is a left vertex} \\ 0 & \text{otherwise} \end{cases}$.
repeat
  Augmentation:
  Find a maximal set P of augmenting paths in $G[1,1]$ and set $M \leftarrow M \oplus P$.
  Dual Adjustment:
  Let F be the left free vertices.
  For all $v \in V_{even}(F, G[1,1])$, set $y(v) \leftarrow y(v) - \delta_0$.
  For all $v \in V_{odd}(F, G[1,1])$ set $y(v) \leftarrow y(v) + \delta_0$.
until $F = \emptyset$ or $y(F) = 0$

After the augmentation step, there will be no augmenting paths in $G[1,1]$, which implies no free vertex is in $V_{odd}(F, G[1,1])$. Therefore, our invariant that right free vertices have zero y-values is maintained after the dual adjustment. Also, since all y-values of free vertices on the left will be decreased in every dual adjustment, they must be minimal among all left vertices. The number of augmentation/dual adjustment steps will be bounded by the number of total possible dual adjustments, which is $2^{\lfloor \log N \rfloor}/\delta_0 \leq 2\sqrt{n}$. Thus, Phase I takes $O(m\sqrt{n})$ time.

In addition, the definition of eligibility on $G[1,1]$ ensures that if there exists $e \in M$ such that $y(e) = w_0(e) + \delta_0$ before the dual adjustment, then $y(e)$ cannot be increased after

the adjustment. Therefore, $y(e) \leq w_0(e) + \delta_0$ for $e \in M$, near tightness is maintained throughout this phase. Likewisely, if $e \notin M$ and $y(e) = w_0(e)$, then $y(e)$ does not decrease during the adjustment. Also, due to the definitions of V_{even} and an alternating path, $y(e)$ does not decrease for all $e \in M$. Thus, $y(e) \geq w_0(e)$ for all $e \in E$, domination is maintained throughtout this phase.

9.2.2 Phase II

At the beginning of scale $i \in [1, L]$, we set $y(u) \leftarrow y(u) + \delta_i$ for all left vertices u and do not change the y-values for all right vertices, so Property 9.2.1(2) (domination) is maintained. So is Property 9.2.1(3) (near tightness):

$$\begin{aligned} y(e) &\leftarrow y(e) + \delta_i \\ &\leq w_{i-1}(e) + \delta_{i-1} + \delta_i && \text{by Property 9.2.1(3) at the end of scale } i-1 \\ &\leq w_i(e) + 3\delta_i && \text{since } \delta_{i-1} = 2\delta_i \text{ and } w_{i-1}(e) \leq w_i(e) \end{aligned}$$

However, Property 9.2.1(4) may be violated, because now the y-values of left free vertices are δ_i. Hence, we will run one iteration of augmentation/dual adjustment step on $G[1, 3]$ described in the pseudocode of Phase I to reduce them to zero. By the same reasoning in Phase I, domination and near tightness ($y(e) \leq w_i(e) + 3\delta_i, \forall e \in M$) will not be violated during the step, which implies Property 9.2.1 is now all maintained.

Next, we will repeat the augmentation/dual adjustment steps described in Section 9.2.2 and 9.2.2 on $G[1, 3]$ until $y(e) \leq w_i(e) + \delta_i$ for all $e \in M$, or equivalently, until $M \cap G[2, 3] = \emptyset$.

There are two reasons that we consider $G[1, 3]$ rather than other definitions for eligibility. First, as in Phase I, since the definition of eligibility does not include matched tight edges, all edges of an augmenting walk become ineligible after we augment along it. Second, when doing the dual adjustment, we will not create any more matched edges in $G[2, 3]$ (though they might be in $G[1, 3]$), since the propagation of dual adjustments along the eligible edges e ensures that $y(e)$ will not be increased for $e \in M \cap G[1, 3]$. This will be explained in Lemma 9.2.8.

Phase II - Augmentation

When augmentation is called in Phase II, we need to eliminate all augmenting walks from the eligibility graph $G[1,3]$. This can be divided into two stages. In the first stage we eliminate the augmenting cycles, whereas in the second stage we eliminate the augmenting paths. Notice that unlike in Phase I, augmenting paths here may start or end with matched edges.

In the first stage, we will find a maximal set of vertex-disjoint augmenting cycles $\mathcal{C}$, which can be done by using a modified depth first search, *cycle_search*(x). We will inflict *cycle_search*(x) on every matched vertex x that has not been visited in previous searches. Recall that x' is the mate of x.

Mark u and u' as visited
for every unmatched edge $u'v$ **do**
  if v is visited and v is an ancestor of u in the search tree **then**
    Add the cycle consisting of the path from v to u' and the edge $u'v$ to $\mathcal{C}$.
    Back up the search until leaving *cycle_earch*(v) so the parent of v is on the top of the stack.
  else if v is not visited **then**
    Call *cycle_search*(v).
  end if
end for

Algorithm 13: *cycle_search*(u)

Lemma 9.2.3. *The algorithm finds a maximal set of vertex-disjoint augmenting cycles $\mathcal{C}$. Moreover, if we augment along every cycle in $\mathcal{C}$, then the graph $G[1,3]$ contains no more augmenting cycles.*

Proof. Suppose the algorithm did not find a maximal set of vertex-disjoint augmenting cycles, let C be such a cycle that is vertex-disjoint from all cycles in $\mathcal{C}$. Let $C = (v_1, v_2, \ldots v_k, v_1)$ so that v_1 is the vertex first entered in the search. Let t be the largest index such that v_t is visited by the search before the search backs up from v_1. Since v_t is not contained in any cycles in $\mathcal{C}$, the search must discover the next vertex of v_t in C. If $t < k$, then v_{t+1} is visited. If $t = k$, then we discovered a cycle containing the edge $v_k v_1$. Both cases lead to a contradiction.

Furthermore, if there exists a cycle C after augmentation, then this cycle must share a vertex v with some cycle $C' \in \mathcal{C}$ due to the maximality of $\mathcal{C}$. However, if v is contained in C, then C contains v and its mate. This contradicts the fact that there will be no eligible matched edge that incidents to v after the augmentation on C'. □

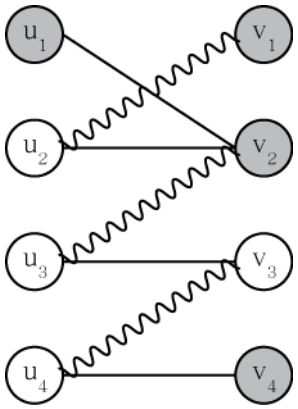


Figure 11: An example illustrating starting vertices and maximal augmenting paths in $G[1,3]$. The plain edges denote unmatched edges, while the curled ones denote matched edges. The shaded vertices denote vertices with zero y-values. Vertex u_1, v_1, and v_2 are starting vertices. The path $P = v_2u_3v_3u_4v_4$ is an augmenting path. However, it is not a maximal augmenting path, since either $u_1v_2u_3v_3u_4v_4$ or $v_1u_2v_2u_3v_3u_4v_4$ is an augmenting path containing P.

In the second stage, we will eliminate all the augmenting paths in $G[1,3]$. This is done by finding a maximal set of vertex-disjoint *maximal augmenting paths*. A maximal augmenting path is an augmenting path that cannot be extended to a longer one (see Figure 11). Note that we require such a path to be maximal, for otherwise it is possible that after we augment along a path, an endpoint of the path becomes free and is now an endpoint of another augmenting path.

Consider the graph $\overrightarrow{G}[1,3]$. It must be a directed acyclic graph, since $G[1,3]$ does not contain an augmenting cycle now. A vertex is said to be a *starting vertex* if it has zero y-value and it is either a left free vertex or a right matched vertex. Therefore, a starting vertex is a possible starting point of an augmenting path in $\overrightarrow{G}[1,3]$. Let S be the set of all starting vertices. We will initiate the search on every unvisited vertex in S in topological order of $\overrightarrow{G}[1,3]$. The way we initiate the search on x depends on whether x is free or not. If x is free,

we will just call $path\_search(x)$. Otherwise, we will call $path\_search(x')$. It is guaranteed that $path\_search(x)$ is called on left vertices.

```
{Recall that x is the starting vertex and 𝒫 is the maximal set of maximal augmenting
paths we have found so far.}
Mark u as visited.
for every unmatched edge uv do
  if v is free {v is a right free vertex} then
    Add the path from x to v to 𝒫 and terminate the search.
  else if v is not visited then
    Call path_search(v').
  end if
end for
if y(u) = 0 {u is a left matched vertex} then
  Add the path from x to u to 𝒫 and terminate the search.
end if
```

Algorithm 14: $path\_search(u)$

If there exists an augmenting walk from x to v and v is not free, our search will explore the possibility that it can be extended from v before it is added to $\mathcal{P}$. If v is free, then it is impossible to extend the path. Furthermore, since we initiated the starting vertices in topological order, it is guaranteed that the path cannot be extended from x either. Therefore, the augmenting path found in our algorithm must be maximal.

Lemma 9.2.4. *After we augment along every path in $\mathcal{P}$, the graph $G[1,3]$ contains no more augmenting paths.*

Proof. Suppose that there exists an augmenting path Q after the augmentation. Then by the maximality of $\mathcal{P}$, there must be some augmenting path in $\mathcal{P}$ sharing vertices with Q. There can be two cases. Case 1: There exists $P \in \mathcal{P}$ and $v \in P \cap Q$ such that v is not an endpoint of either P or Q. In this case, by our definition of an augmenting path, P contains v and its mate before the augmentation on P and Q contains v and its mate after the augmentation. However, after the augmentation on P, there should be no eligible matched edge that incidents to v, thus Q cannot contain both v and its mate. Case 2: For all $P \in \mathcal{P}$, either $Q \cap P = \emptyset$ or Q and P intersect on their endpoints. Let P be the earliest path added to $\mathcal{P}$ such that P and Q intersect. Let x be the endpoint where they intersect, and x_P and

x_Q be the other endpoints of P and Q. If $x_P = x_Q$ then there was an augmenting cycle, which is not possible. If x_P is a starting vertex, then *path_search*(x_P) should have found a longer augmenting path than P, since PQ is a longer one. On the other hand, if x is a starting vertex, it must be a right matched vertex and becomes free after augmentation, so x_Q must also be a starting vertex. Since our search is called in topological order on starting vertices, x_Q must be called before x, which implies that the first augmenting path found that intersects Q contains x_Q but not x. □

Phase II - Dual Adjustment

Let B be the set of violated matched edges that need to be tightened before the end of scale i, that is, $B = \{e \in M : y(e) - w_i(e) > \delta_i\} = G[2,3] \cap M$. Define the *badness*, $f(e)$, to be the amount edge e has violated. That is, $f(e)$ is $(y(e) - w_i(e) - \delta_i)/\delta_i$ for $e \in B$, $f(e)$ is 0 for $e \notin B$. Let $f(B) = \sum_{e \in B} f(e)$ be the total badness of B. Then B is empty if and only if $f(B) = 0$, since $f(e) > 0$ for $e \in B$. The goal of dual adjustment is to tighten Property 9.2.1(3), namely, to decrease $f(B)$ to 0.

A $B' \subseteq B$ is said to be a *chain* if there is an eligible alternating path containing B'. On the other hand, B' is said to be an *anti-chain* if for any $m_1, m_2 \in B'$ such that $m_1 \neq m_2$, there exists no eligible alternating path containing them.

Lemma 9.2.5. *For any $t > 1$, there exists $B' \subseteq B$ such that either B' is a chain with $f(B') \geq \lceil t \rceil$ or B' is an anti-chain with $|B'| \geq \lceil f(B)/2t \rceil$. Moreover, such B' can be found in linear time.*

Proof. This lemma basically follows from Dilworth's Lemma [34]. First obtain $\overrightarrow{G}[1,3]$ by orienting the edges in $G[1,3]$ and assign the length to be $f(e)$ for every $e \in \overrightarrow{G}[1,3]$. Then, $\overrightarrow{G}[1,3]$ must be a directed acyclic graph since we have no augmenting cycles.

Let S denote the vertices with zero in-degrees. Compute the longest path from S to every vertex in $\overrightarrow{G}[1,3]$, which can be done in linear time in topological order. If there exists a path P having length at least $\lceil t \rceil$, then $P \cap B$ must be a chain with $f(P \cap B) \geq \lceil t \rceil$. Otherwise, for every $uv \in B$ (assume that v is the left vertex), the length of the longest path from S to v is in the range of $[1, \lceil t \rceil - 1]$. Since $f(e) \leq 2$ for $e \in B$, we must have at least

$\lceil |B|/t \rceil \geq \lceil f(B)/2t \rceil$ such v having the same longest distance from S. If the distance is k, then the set $B' = \{uv \in B : v \text{ is a left vertex and the longest distance from } S \text{ to } v \text{ is } k\}$ must be an anti-chain. For $u_1v_1, u_2v_2 \in B$, if there is an alternating path containing them in $G[1,3]$, there must be a path from u_1v_1 to u_2v_2 or from u_2v_2 to u_1v_1 in $\overrightarrow{G}[1,3]$ so the longest distance from S to v_1 and v_2 must be different. □

Below we show that if B' is a chain we can decrease the total badness by $f(B')$ in linear time. On the other hand, if B' is an anti-chain, then we can decrease the total badness by $|B'|/2$ also in linear time.

Phase II - Dual Adjustment - Anti-chain Case

Definition 9.2.6. *A vertex x is said to be dual adjustable if for every $v \in V_{odd}(x, G[1,3])$, v is not free and for every $v \in V_{even}(x, G[1,3])$, $y(v) > 0$.*

Lemma 9.2.7. *For every $e = uv \in B$, either u is adjustable or v is adjustable or both. Furthermore, all adjustable vertices can be found in $O(m)$ time.*

Proof. First, if $e = uv \in B$ and u and v are both not adjustable, then by our definition of adjustable, there exist vertices w and x having zero y-values where $w \rightsquigarrow u \to v \rightsquigarrow x$ is an augmenting path. However, this contradicts the fact that there are no augmenting paths after the augmentation step.

To find the adjustable vertices, it is rather convenient to mark up all those unadjustable vertices. Let $\tilde{V} = \{v : v \text{ is free or } v \text{ is matched and } y(v') = 0\}$, and mark all vertices as unadjustable in $V_{odd}(\tilde{V}, G[1,3])$. This can be done in linear time. □

Let $B' \subseteq B$ be an anti-chain. We call *antichain_adjust*(B') to adjust the dual variables. In the procedure, we will pick a set of dual adjustable vertices X that are adjacent to B' and on the same side, then do a dual adjustment starting at X. Since by Lemma 9.2.7, for any $e = uv \in B'$ either u is adjustable or v is adjustable or both, we can guarantee that $|X| \geq |B'|/2$. See Figure 12 for an example.

Lemma 9.2.8. *The dual adjustment starting at X will not break Property 9.2.1(1), 9.2.1(2), or 9.2.1(4). Furthermore, it makes Property 9.2.1(3) tighter by decreasing $f(B)$ by $|X|$.*

Let $\tilde{V} = \{v : v \text{ is free or } v \text{ is matched and } y(v') = 0\}$.
Mark vertices in $V \setminus V_{odd}(\tilde{V}, G[1,3])$ as adjustable vertices.
Let $X_L = \{u : uv \in B' \text{ and } u \text{ is a left adjustable vertex}\}$,
$X_R = \{u : uv \in B' \text{ and } u \text{ is a right adjustable vertex}\}$.
If $|X_R| > |X_L|$, then let $X = X_R$; otherwise let $X = X_L$.
Dual adjustment starting at X:
For all $v \in V_{even}(X, G[1,3])$, set $y(v) \leftarrow y(v) - \delta_i$.
For all $v \in V_{odd}(X, G[1,3])$, set $y(v) \leftarrow y(v) + \delta_i$.

Algorithm 15: *antichain_adjust*(B')

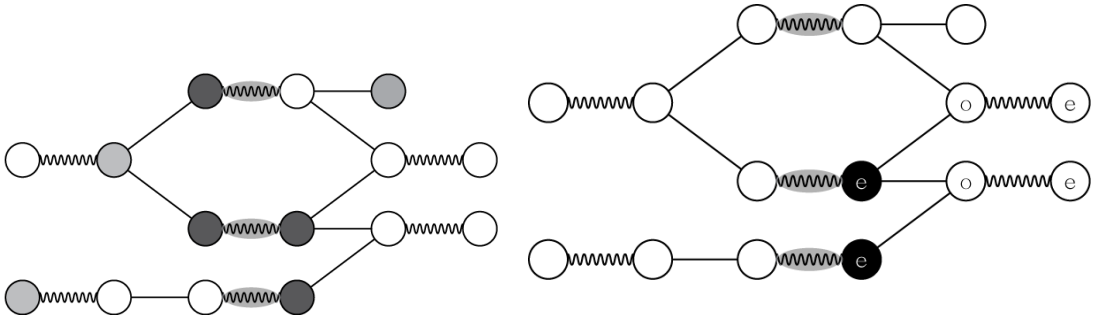


Figure 12: An example of an eligible graph that illustrates an anti-chain and adjustable vertices. (a) The light shaded vertices denote vertices with zero y-values. The shaded matched edges form an anti-chain of size 3. The dark shaded vertices are adjustable vertices of the anti-chain. (b) The dark vertices denote X, the selected vertices for the dual adjustment. Vertices marked with 'e' and 'o' denote vertices in $V_{even}(X, G[1,3])$ and $V_{odd}(X, G[1,3])$ respectively.

Proof. Since every vertex in X is adjustable, every vertex $v \in V_{odd}(X, G[1,3])$ must have $y(v) > 0$, implying $y(v)$ will be non-negative after subtracting δ_i. Thus, Property 9.2.1(1) is maintained. In addition, by the definitions of an alternating path and V_{even}, $V_{even}(X, G[1,3])$ cannot contain a free vertex. Therefore, no dual variables of free vertices are adjusted, meaning Property 9.2.1(4) is maintained. Since all vertices in X are on the same side, $y(e)$ can change by at most δ_i. We only need to check:

1. If $e = uv$ is tight before the adjustment, Property 9.2.1(2) (domination) holds for e after the adjustment: If $e \notin M$, then e is eligible. If the y-value of an endpoint gets subtracted by δ_i then another endpoint must be added by δ_i, which means $y(e)$ does not decrease. If $e \in M$, then it is not possible for u or v to be in $V_{even}(x, G[1,3])$, since

e is ineligible and we start with an unmatched edge. Therefore, domination holds on e after the adjustment.

2. $f(B)$ decreases by $|X|$: If e is tight before the adjustment, then increasing $y(e)$ by δ_i contributes nothing to $f(B)$. If e is not tight, then e is eligible and $f(e)$ cannot be increased either, since if one endpoint gets added by δ_i, then another endpoint must be subtracted by δ_i. Furthermore, if $e \in B'$ and e is incident to a vertex in X, then one endpoint of e is in $V_{even}(X, G[1,3])$ and the other cannot be in $V_{odd}(X, G[1,3])$, because B' is an anti-chain. Therefore, $f(e)$ decreases by exactly 1.

□

Therefore, by doing the dual adjustment starting at X, we can decrease $f(B)$ by at least $|B'|/2$.

Phase II - Dual Adjustment - Chain Case

In the chain case, there exists an alternating path containing B'. Take P to be the minimal alternating path containing B' so that P starts and ends with edges in B'. If we augment along P, then the edges in B' no longer contribute to $f(B)$ since they become unmatched, and new M-edges contribute nothing to $f(B)$. However, the endpoints of P, say u and v, become free while possibly having positive y-values. Hence we will need to make them matched by augmentation or decrease their y-values to zero. In this subsection, we relax our definition of augmenting path such that the y-value of each endpoint is 0 **except** if it is u or v. We perform a search similar to Phase I on u until an augmenting path P_u starting from u is found or $y(u)$ becomes zero (which is a degenerated case when $P_u = \{u\}$). After the search, we will not augment P_u immediately but perform another search again on v to find an augmenting path P_v. Now if there exists an augmenting path Q in $G[0,3]$ whose endpoints are u and v, then we will augment along it. See Figure 13 for an example. Otherwise, we let $Q = P_u \cup P_v$ and then augment along Q. In this case, we must have $P_u \cap P_v = \emptyset$, for otherwise an augmenting path in $G[0,3]$ between u and v exists. In the searches, we will use $G[0,3]$ as the eligibility graph, which ensures the weight of the new matching we get does not decrease. Below we describe how the search works.

Let $x \in \{u, v\}$ be the free vertex that we perform the search on. If there exists an augmenting path in $G[0,3]$ starting at x, then we will stop. Recall that the other endpoint of x can be either free or matched. On the other hand, if there is no augmenting path, then let γ be the minimum of $min\{y(z) : z \in V_{even}(x, G[0,3])\}$ and $min\{y(v_1v_2) - w_i(v_1v_2) : v_1 \in V_{even}(x, G[0,3])$ and $v_2 \notin V_{odd}(x, G[0,3])\}$. Then, add γ to the y-value of every vertex in $V_{odd}(x, G[0,3])$ and subtract γ from vertices in $V_{even}(x, G[0,3])$. Keep repeating the adjustment until we find an augmenting path starting at x. Similar to one iteration in the Hungarian algorithm, this process is equivalent to computing shortest paths from x, which is described in Algorithm 16, $search(x)$.

If x is a right vertex, set $\overrightarrow{G} \leftarrow \overrightarrow{G}^T$ (reverse the edges).

For each $e \in \overrightarrow{G}$, assign a new weight $w'(e) = \begin{cases} y(e) - w_i(e) & \text{if } e \notin M \\ 0 & \text{if } e \in M \end{cases}$

Compute the distance $d(z)$ from x to z for every $z \in \overrightarrow{G}$, where $d(z) = \infty$ if z is not reachable from x.

Let $h(z) = \begin{cases} d(z) & \text{if } z \text{ is free and not on the same side as } x \\ d(z) + y(z) & \text{if } z \text{ is on the same side as } x \\ \infty & \text{otherwise} \end{cases}$.

Let z_{min} be the vertex such that $h(z)$ is minimum, and let $\Delta = h(z_{min})$.

Let P_x be the shortest path from x to z_{min}.

Set $y(z) \leftarrow \begin{cases} y(z) - max\{0, \Delta - d(z)\} & \text{if } z \text{ is on the same side as } x \\ y(z) + max\{0, \Delta - d(z)\} & \text{if } z \text{ is not on the same side as } x \end{cases}$

return P_x

Algorithm 16: $search(x)$

In $search(x)$, Δ is the amount of dual adjustment needed before an augmenting path opens up. The augmenting path starts from x and ends at some z_{min}, where z_{min} can be either a free vertex on the opposite side of x or a zero y-valued matched vertex on the same side as x. For the former situation, the dual adjustment needed is $d(z_{min})$. For the latter situation, we not only need to reach z_{min} but also need to decrease its y-value to 0, so the dual adjustment needed is $d(z_{min}) + y(z_{min})$. After finding Δ, we will adjust the dual variables accordingly. $search(x)$ returns an augmenting path P_x starting at x.

By Property 9.2.1(1), $d(z)$ must be a non-negative multiple of δ_i. Since our goal of computing the shortest path is to find Δ, we can just compute those $d(z)$ which are no more than Δ.

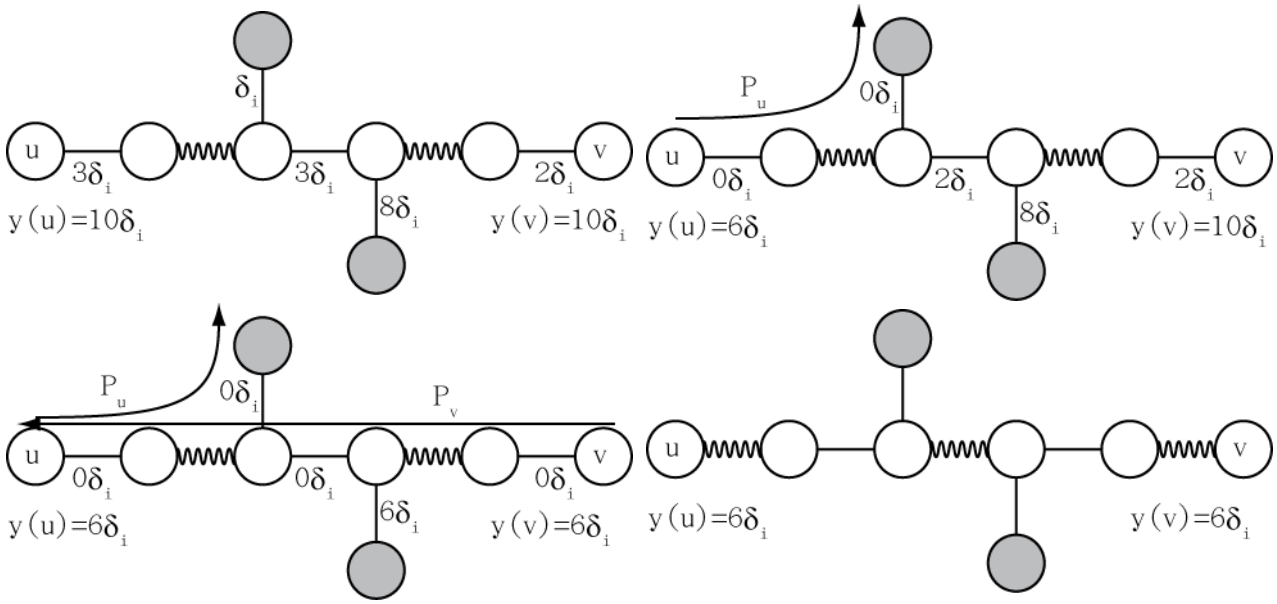


Figure 13: An example illustrating procedures for the chain case. Edges are shown with their new weight w'. The shaded vertices are free vertices with zero y-values. (a) After augmenting along P, u and v became free while having positive y-values. (b) $search(u)$ adjusted $\Delta_u = 4\delta_i$ and found an augmenting path P_u. (c) $search(v)$ adjusted $\Delta_v = 4\delta_i$ and found P_v. (d) Augmentation along Q. This is the case where there exists an augmenting path Q between u and v in $G[0,3]$, which happens to be P_v in the example.

This can be done in $O(m + \Delta/\delta_i)$ time by using an array as a priority queue in Dijkstra's algorithm. (See Dial's implementation [32].)

Lemma 9.2.9. *Augmenting along P and then Q does not decrease the weight of the matching and $\Delta_u + \Delta_v \leq 3n\delta_i$, where Δ_u and Δ_v are the amount of dual adjustments done in $search(u)$ and $search(v)$. Thus, the search can be done in $O(m)$ time.*

Proof. Suppose M is the original matching, M' is the matching obtained by augmenting along P, and M'' is the final matching after augmenting along Q. Let $w''(e) = y(e) - w_i(e)$ (notice that w'' differs from w' on the matched edges). For a quantity q denote its value before both searches by q_{old} and after both searches by q_{new}. After the searches, we must

have:

$$\begin{aligned} w_i(Q \setminus M') &= \sum_{e \in Q \setminus M'} y_{new}(e) && \text{tightness on unmatched edges} \\ &= y_{new}(u) + y_{new}(v) + \sum_{e \in M' \cap Q} y_{new}(e) && (*) \\ &= y_{new}(u) + y_{new}(v) + w_i(M' \cap Q) + w''_{new}(Q \cap M') && \text{defn. of } w''_{new} \end{aligned}$$

Therefore,

$$w_i(M'') = w_i(M') + y_{new}(u) + y_{new}(v) + w''_{new}(Q \cap M') \tag{9.3}$$

(*) holds because beside u and v, the other possible difference of vertices in $Q \setminus M'$ and $Q \cap M'$ are those with zero y-values, which are the endpoints of P_u and P_v when $Q = P_u \cup P_v$.

Similarly, before the searches, we have:

$$w_i(M) = w_i(M') + y_{old}(u) + y_{old}(v) - w''_{old}(P \setminus M') \tag{9.4}$$

The amount of dual adjustments is at most the distance between u and v, so $\Delta_u + \Delta_v \leq w''_{old}(P \setminus M') \leq 3n\delta_i$. Moreover:

$$\begin{aligned} w_i(M'') &\geq w_i(M') + y_{new}(u) + y_{new}(v) && \text{by (9.3) and } w''_{new}(Q \cap M') \geq 0 \\ &= w_i(M') + y_{old}(u) + y_{old}(v) - \Delta_u - \Delta_v \\ &\geq w_i(M') + y_{old}(u) + y_{old}(v) - w''_{old}(P \setminus M') \\ &= w_i(M) && \text{by (9.4)} \end{aligned}$$

□

Lemma 9.2.10. *At most $O(\sqrt{n})$ rounds of augmentation and dual adjustment are required to reduce $f(B)$ to 0.*

Proof. When $f(B) = b$, choose $t = \sqrt{b}/2$. Either we can obtain an anti-chain B' of size at least $\lceil\sqrt{b}\rceil$ and decrease $f(B)$ by $\lceil\sqrt{b}/2\rceil$, or we can obtain a chain B' such that $f(B') \geq \lceil\sqrt{b}/2\rceil$ and decrease $f(B)$ by $f(B')$. In any case, we can decrease $f(B)$ by $\lceil\sqrt{b}/2\rceil$. The

number of rounds is at most $T(b)$, where $T(b) = T(b - \lceil\sqrt{b}/2\rceil) + 1$ for $b > 0$ and $T(b) = 0$ for $b = 0$. It can be shown by induction that $T(b) \leq 4\sqrt{b}$, so that $T(b) \leq 4\sqrt{b} \leq 4\sqrt{2n}$. □

9.2.3 Phase III

The procedure for Phase III is similar to that for Phase II, but with several differences. First, instead of operating on $G[1, 3]$, we will operate on $G[0, 1]$ in this phase. Second, in the augmentation step, the definition of augmenting walks is modified such that the walk must contain at least one matched edge that is not tight. One exception is that an augmenting path in $G[0, 3]$ of the chain case still refers to the old definition in Phase II, where we do not require it to contain at least one non-tight edge. Third, the way we find augmenting walks will be different from Phase II, since a tight edge in an augmenting walk will not become ineligible after an augmentation.

Phase III - Augmentation

Lemma 9.2.11. *Each augmentation along the augmenting walk in $G[0, 1]$ increases the weight of M. Consequently, there can be at most $\sqrt{n}$ augmenting walks in Phase III.*

Proof. Let M be the original matching and M' be the matching after augmentation. Suppose P is an augmenting walk. We must have $\sum_{v\in M\cap P} y(v) = \sum_{v\in M'\cap P} y(v)$, regardless of whether P is an augmenting cycle or an augmenting path. Since P contains at least one non-tight matched edge, $w(M \cap P) < \sum_{v\in M\cap P} y(v) = \sum_{v\in M'\cap P} y(v) = w(M' \cap P)$. Since all weights are integers, the weight of the matching is increased by at least one.

After Phase II, $\delta_L = 2^{\lfloor \log N/\sqrt{n}\rfloor - \lceil \log N\rceil} \leq 1/\sqrt{n}$. By Lemma 9.2.2, we have $w(M) \geq w(M^*) - n\delta_L \geq w(M^*) - \sqrt{n}$. Since each augmentation increases the weight of M by at least one, and by Lemma 9.2.9, the weight of M does not decrease in dual adjustment steps, there can be at most $\sqrt{n}$ augmentations. □

We have to ensure that no augmenting walks exist in $G[0, 1]$ after the augmentation step. Since an augmenting walk may contain tight matched edges in $G[0, 1]$, Lemma 9.2.3 and

Lemma 9.2.4 no longer guarantee augmenting along a maximal set of augmenting cycles/paths will break up all eligible cycles/paths. However, by Lemma 9.2.11, we only need to find one augmenting walk in time $O(m)$ if it exists.

This can be done by the following procedure. First, obtain $\overrightarrow{G}[0,1]$ by orienting the edges of $G[0,1]$. If there is an augmenting cycle, then the cycle must contain a non-tight edge, say e. Also, the endpoints of e must be strongly connected in $\overrightarrow{G}[0,1]$. Therefore, to detect such cycles, run a strongly connected component algorithm first and then check whether the endpoints of non-tight edges are strongly connected.

Second, to detect an augmenting path, run the algorithm in $O(m)$ time described in Lemma 9.2.7 to determine whether v is adjustable for all $v \in G$. If there exists a non-tight matched edge uv such that both u and v are not adjustable, then there must be an augmenting path containing uv. Therefore, an augmenting walk can be found in $O(m)$ time, if one exists, and the total time spent on augmentation during Phase III is $O(m\sqrt{n})$.

Phase III - Dual Adjustment

In Phase III, our goal is to tighten all non-tight edges. Thus, these edges are considered to be violated. That is, $B = \{e \in M : y(e) - w_i(e) = \delta_i\} = G[1,1] \cap M$. The definition of badness, $f(e)$, is changed to $(y(e) - w_i(e))/\delta_i$ accordingly. In Lemma 9.2.7, if u and v are both unadjustable, it is true that there will be an augmenting path containing a non-tight edge, which is uv. This will contradict with the fact that there are no augmenting paths after the augmentation step. Therefore, the lemma still works.

The other difference is Lemma 9.2.5, where the graph $\overrightarrow{G}$ may contain zero-length cycles or zero-length paths now. However, that does not affect how we select a chain or an anti-chain. The difference is that the graph may not be a DAG now but we still need to compute the length of the longest path from S to every vertex in linear time, where S is the set of vertices with zero in-degrees. This can be done by the following procedure:

1. Find the strongly connected components of $\overrightarrow{G}[0,0]$.
2. For each strongly connected component C in $\overrightarrow{G}[0,0]$, contract C into one vertex in $\overrightarrow{G}[0,1]$, because all vertices in C are supposed to have the same length of longest path from S in $\overrightarrow{G}[0,1]$.

3. Compute the longest path in the new contracted graph, which is a DAG.

Lemma 9.2.8, Lemma 9.2.9, and Lemma 9.2.10 all hold if we replace $G[1,3]$ by $G[0,1]$, and the existence of tight augmenting cycles/paths does not affect their correctness. Thus, the total time spent on dual adjustment in Phase III for $f(B)$ to reach zero is still $O(m\sqrt{n})$.

9.2.4 Maximum Weighted Perfect Matching

Suppose a perfect matching exists in G. By the reduction described in Section 9.1, where we added nN weight to every edge, we can solve the MWPM problem in $O(m\sqrt{n}\log(nN))$ time. This does not improve the previous bound in [60]. However, below we give an algorithm that uses fewer scales, $\lceil\log(\sqrt{n}N)\rceil$ instead of $\lceil\log(nN)\rceil$. This is done by modifying the algorithm for MWM, where we maintain the following:

Property 9.2.12. *Let $\delta_0 = 2^{\lfloor\log N\rfloor}$, $L = \lceil\log(\sqrt{n}N)\rceil$. At the end of each scale $i \in [0, L]$, we maintain a perfect matching M with the following:*

1. ***(Granularity of** y**)** $y(u)$ *is a multiple of* δ_i.
2. ***(Domination)*** $y(e) \geq w_i(e)$ *for all* $e \in E$.
3. ***(Near Tightness)*** $y(e) \leq w_i(e) + 3\delta_i$. *At the end of scale i, it is tightened so that* $y(e) \leq w_i(e) + \delta_i$,

For scale $i = 0$, find a perfect matching M using the Hopcroft-Karp algorithm in $O(m\sqrt{n})$ time [83], and assign $y(u) \leftarrow \delta_0$ to all left vertices u, $y(v) \leftarrow 0$ to all right vertices v.

Next, begin Phase II for scale $i \in [1, L]$ and Phase III at the end of scale L with the following modifications:

1. An augmenting walk only refers to an alternating cycle. We no longer consider augmenting paths so that we always keep the matching M to be perfect. More precisely, in the augmentation step, we no longer run *path_search*(x). In the dual adjustment step, if it is the anti-chain case, then either side of an edge in B' is adjustable, since

now we allow the dual variables to have negative values. Therefore, for an anti-chain B' we can always decrease $f(B)$ by $|B'|$.

In the chain case, the endpoints of P, say u and v, are freed temporarily. Here, we must force the $search(x)$ to find an augmenting path to connect u and v back. This can be done by only doing $search(u)$ until the augmenting path in $G[0,3]$ between u and v opens up (e.g. force z_{min} to be v), which is always possible since we no longer need to keep y-values non-negative.

2. When $f(B) = b$, choose $t = \sqrt{b/2}$. Either we can obtain an anti-chain B' of size at least $\lceil\sqrt{b/2}\rceil$ and decrease $f(B)$ by $\lceil\sqrt{b/2}\rceil$, or we can obtain a chain B' such that $f(B') \geq \lceil\sqrt{b/2}\rceil$ and decrease $f(B)$ by $f(B')$. In any case, we can decrease $f(B)$ by $\lceil\sqrt{b/2}\rceil$, so the number of rounds is at most $T(b) = T(b - \lceil\sqrt{b}/2\rceil) + 1$. It can be shown by induction, $T(b) \leq 2\sqrt{2b} \leq 4\sqrt{n}$.

3. When Phase II ends at scale L, the result of Lemma 9.2.11 also holds. Since $\delta_L = 2^{\lfloor \log N \rfloor - \lceil \log(\sqrt{n}N) \rceil} \leq 1/\sqrt{n}$, by Lemma 9.2.2, $w(M) \geq w(M^*) - n\delta_L \geq w(M^*) - \sqrt{n}$. Thus, the matching can be improved at most $\sqrt{n}$ times.

Therefore, the algorithm runs in $\lceil \log(\sqrt{n}N) \rceil$ scales, where each scale takes $O(m\sqrt{n})$ time. The reason why we cannot achieve the same bound as the MWM problem is because Phase I does not apply. It is still unknown whether the MWPM problem can be solved in $O(m\sqrt{n}\log N)$ time.

9.3 Discussion

We believe that finding the MWM is easier than finding the MWPM. In Chaper 8, we gave a scaling algorithm that solves the MWPM problem on general graphs in $O(m\sqrt{n}\log(nN))$ time. It is an interesting open problem whether the MWM on general graphs can be found also in $O(m\sqrt{n}\log N)$ time.

There are some reasons why it seems not possible to extend this algorithm directly to the general graph case, where there are blossoms. When we find an augmenting walk passing a blossom node, the edges inside the blossom become ineligible. This no longer guarantees that

augmenting along the next augmenting walk passing this blossom will increase the weight of the matching, which makes Lemma 9.2.11 inapplicable.

APPENDICES

Appendix A

Tools

Lemma A.1. *$e^{-x} \leq 1 - x/2$ for $0 \leq x \leq 1.59$.*

Proof. Let $f(x) = e^{-x} - 1 + x/2$. $f(0) = 0$ and $f(1.59) \leq 0$. $f'(x) = 1/2 - e^{-x}$, $f'(x)$ is zero only when $x = \ln 2 \leq 1.59$. Since $f(\ln 2) \leq 0$, $f(x) \leq 0$ for $0 \leq x \leq 1.59$. □

The following lemma shows when conditioning on a very likely event B, the probability of an event can only be affected by a small amount.

Lemma A.2. *Let A, B be two events, $|\Pr(A) - \Pr(A|B)| \leq \Pr(\overline{B})$.*

Proof. $\Pr(A) = \Pr(B)\Pr(A|B) + \Pr(\overline{B})\Pr(A|\overline{B}) = \Pr(A|B) + \Pr(\overline{B})(\Pr(A|\overline{B}) - \Pr(A|B))$. Therefore, $|\Pr(A) - \Pr(A|B)| \leq \Pr(\overline{B})$. □

See Dubhashi and Panconesi [40] for proofs of Lemma A.3, Lemma A.4, and related concentration bounds.

Lemma A.3. (Hoeffding's Inequaliy) *Let $X_1, \ldots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$ for $1 \leq i \leq n$. Let $X = \sum_i X_i$, then for any $t > 0$,*

$$\Pr(X > \mathrm{E}[X] + t) \leq e^{-\frac{2t^2}{\sum_i (b_i - a_i)^2}}$$

Lemma A.4. (Chernoff Bound) *Let $X_1, \ldots, X_n$ be indicator variables such that $\Pr(X_i = 1) = p$. Let $X = \sum_{i=1}^{n} X_i$. Then, for $\delta > 0$:*

$$\Pr(X \geq (1+\delta)\,\mathrm{E}[X]) < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mathrm{E}[X]}$$
$$\Pr(X \leq (1-\delta)\,\mathrm{E}[X]) < \left[\frac{e^{\delta}}{(1-\delta)^{(1-\delta)}}\right]^{\mathrm{E}[X]}$$

The two bounds above imply that for $0 < \delta < 1$, we have:

$$\Pr(X \geq (1+\delta)\,\mathrm{E}[X]) < e^{-\delta^2\,\mathrm{E}[X]/3}$$
$$\Pr(X \leq (1-\delta)\,\mathrm{E}[X]) < e^{-\delta^2\,\mathrm{E}[X]/2}.$$

Corollary A.1. *Let $X_1, \ldots, X_n$ be indicator variables such that $\Pr(X_i) = p_i$. Let $X = \sum_{i=1}^{n} X_i$. If $M \geq \mathrm{E}[X]$ and $0 < \delta \leq 1$, then*

$$\Pr(X > \mathrm{E}[X] + \delta M) \leq e^{-\delta^2 M/3}$$

Proof. Without loss of generality, assume $M = t\,\mathrm{E}[X]$ for some $t \geq 1$, we have

$$\begin{aligned}
\Pr(X > \mathrm{E}[X] + \delta M) &\leq \left[\frac{e^{t\delta}}{(1+t\delta)^{(1+t\delta)}}\right]^{\mathrm{E}[X]} && \text{by Lemma A.4} \\
&= \left[\frac{e^{\delta}}{(1+t\delta)^{(1+t\delta)/t}}\right]^{M} \\
&\leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{M} && (*) \\
&\leq e^{-\delta^2 M/3} && \tfrac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \leq e^{-\delta^2/3} \text{ for } 0 < \delta < 1
\end{aligned}$$

Inequality (*) follows if $(1+t\delta)^{(1+t\delta)/t} \geq (1+\delta)^{(1+\delta)}$, or equivalently, $((1+t\delta)/t)\ln(1+t\delta) \geq (1+\delta)\ln(1+\delta)$. Letting $f(t) = ((1+t\delta)/t)\ln(1+t\delta) - (1+\delta)\ln(1+\delta)$, we have $f'(t) = \frac{1}{t^2}(\delta t - \ln(1+\delta t)) \geq 0$ for $t > 0$. Since $f(1) = 0$ and $f'(t) \geq 0$ for $t > 0$, we must have $f(t) \geq 0$ for $t \geq 1$. □

Lemma A.5. *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be (likely) events and $X_1, \ldots, X_n$ be indicator variables such*

that for each $1 \le i \le n$ *and* $X = \sum_{i=1}^{n} X_i$,

$$\max_{\boldsymbol{X}_{i-1}} \Pr(X_i \mid \boldsymbol{X}_{i-1}, \mathcal{E}_1, \ldots \mathcal{E}_i) \le p$$

where $\boldsymbol{X}_i$ *denotes the shorthand for* $(X_1, \ldots, X_i)$.[1] *Then for* $\delta > 0$*:*

$$\Pr\left((X > (1+\delta)np) \cap \left(\bigcap_i \mathcal{E}_i\right)\right) \le \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^{np}$$

and thus by the union bound,

$$\Pr(X > (1+\delta)np) \le \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^{np} + \sum_i \Pr(\overline{\mathcal{E}_i}).$$

Proof. For now let us treat $\mathcal{E}_i$ as 0/1 random variables and let $\mathcal{E} = \prod_i \mathcal{E}_i$. For any $t > 0$,

$$\begin{aligned}
\Pr\left((X > (1+\delta)np) \cap \left(\bigcap_i \mathcal{E}_i\right)\right) &= \Pr\left(\left(\prod_{i=1}^{n} \mathcal{E}_i\right) \cdot \exp(tX) > \exp(t(1+\delta)np)\right) \\
&\le \frac{\mathrm{E}[(\prod_{i=1}^{n} \mathcal{E}_i) \cdot \exp(tX)]}{\exp(t(1+\delta)np)} \\
&= \frac{\mathrm{E}[(\prod_{i=1}^{n} \mathcal{E}_i \cdot \exp(tX_i))]}{\exp(t(1+\delta)np)} \qquad \text{(A.1)}
\end{aligned}$$

We will show by induction that

$$\mathrm{E}\left[\left(\prod_{i=1}^{k} \mathcal{E}_i \exp(tX_i)\right)\right] \le (1 + p(e^t - 1))^k$$

[1] We slightly abuse the notation that when conditioning on the random variable X_i, it means X_i may take arbitrary values, whereas when conditing on the event $\mathcal{E}_i$, it means that $\mathcal{E}_i$ happens.

When $k = 0$, it is trivial that $\mathrm{E}[\mathcal{E}] \leq 1$.

$$\begin{aligned}
\mathrm{E}\left[\left(\prod_{i=1}^{k} \mathcal{E}_i \exp(tX_i)\right)\right] &\leq \mathrm{E}\left[\left(\prod_{i=1}^{k-1} \mathcal{E}_i \exp(tX_i)\right) \cdot \mathrm{E}\left[\mathcal{E}_k \exp(tX_k) \mid \boldsymbol{X_{i-1}}, \mathcal{E}_1, \ldots, \mathcal{E}_{k-1}\right]\right] \\
&= \mathrm{E}\left[\left(\prod_{i=1}^{k-1} \mathcal{E}_i \exp(tX_i)\right) \cdot \Pr(\mathcal{E}_k) \cdot \mathrm{E}\left[\exp(tX_k) \mid \boldsymbol{X_{i-1}}, \mathcal{E}_1, \ldots, \mathcal{E}_k\right]\right] \\
&\leq \mathrm{E}\left[\left(\prod_{i=1}^{k-1} \mathcal{E}_i \exp(tX_i)\right) \cdot \mathrm{E}\left[\exp(tX_k) \mid \boldsymbol{X_{i-1}}, \mathcal{E}_1, \ldots, \mathcal{E}_k\right]\right] \\
&= \mathrm{E}\left[\left(\prod_{i=1}^{k-1} \mathcal{E}_i \exp(tX_i)\right) \cdot (1 + \Pr(X_k \mid \boldsymbol{X}_{i-1}, \mathcal{E}_1, \ldots, \mathcal{E}_k)(e^t - 1))\right] \\
&\leq \mathrm{E}\left[\left(\prod_{i=1}^{k-1} \mathcal{E}_i \exp(tX_i)\right) \cdot (1 + p(e^t - 1))\right] \\
&= \mathrm{E}\left[\left(\prod_{i=1}^{k-1} \mathcal{E}_i \exp(tX_i)\right)\right] \cdot (1 + p(e^t - 1)) \\
&\leq (1 + p(e^t - 1))^k
\end{aligned}$$

Therefore, by (A.1),

$$\begin{aligned}
\Pr\left((X > (1+\delta)np) \cap \left(\bigcap_i \mathcal{E}_i\right)\right) &= \frac{\mathrm{E}[\mathcal{E} \cdot \prod_{i=1}^{n} \exp(tX_i)]}{\exp(t(1+\delta)np)} \\
&\leq \frac{(1 - p(e^t - 1))^n}{\exp(t(1+\delta)np)} \\
&\leq \frac{\exp(np(e^t - 1))}{\exp(t(1+\delta)np)} \\
&= \left[\frac{\exp(\delta)}{(1+\delta)^{1+\delta}}\right]^{np}.
\end{aligned}$$

The last equality follows from the standard derivation of Chernoff Bound by choosing $t = \ln(1 + \delta)$. □

Corollary A.2. *Suppose that for any $\delta > 0$,*

$$\Pr\left((X > (1+\delta)np) \cap \left(\bigcap_i \mathcal{E}_i\right)\right) \leq \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^{np}$$

then for any $M \geq np$ *and* $0 < \delta < 1$,

$$\begin{aligned}\Pr\left((X > np + \delta M) \cap \left(\bigcap_i \mathcal{E}_i\right)\right) &\leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^M \\ &\leq e^{-\delta^2 M/3}\end{aligned}$$

Proof. Without loss of generality, assume $M = tnp$ for some $t \geq 1$, we have

$$\begin{aligned}&\Pr\left((X > np + \delta M) \cap \left(\bigcap_i \mathcal{E}_i\right)\right) \\ &\leq \left[\frac{e^{t\delta}}{(1+t\delta)^{(1+t\delta)}}\right]^{np} \\ &= \left[\frac{e^{\delta}}{(1+t\delta)^{(1+t\delta)/t}}\right]^M \\ &\leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^M \qquad (*) \\ &\leq e^{-\delta^2 M/3} \qquad \frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \leq e^{-\delta^2/3} \text{ for } 0 < \delta < 1\end{aligned}$$

Inequality (*) follows if $(1+t\delta)^{(1+t\delta)/t} \geq (1+\delta)^{(1+\delta)}$, or equivalently, $((1+t\delta)/t)\ln(1+t\delta) \geq (1+\delta)\ln(1+\delta)$. Letting $f(t) = ((1+t\delta)/t)\ln(1+t\delta) - (1+\delta)\ln(1+\delta)$, we have $f'(t) = \frac{1}{t^2}(\delta t - \ln(1+\delta t)) \geq 0$ for $t > 0$. Since $f(1) = 0$ and $f'(t) \geq 0$ for $t > 0$, we must have $f(t) \geq 0$ for $t \geq 1$. □

Lemma A.6 ([40], Azuma's inequality)**.** *Let* f *be a function of* n *random variables* $X_1, \ldots, X_n$ *such that for each* i*, any* $\boldsymbol{X}_{i-1}$*, any* a_i *and* a_i'*,*

$$|\mathrm{E}[f \mid \boldsymbol{X}_{i-1}, X_i = a_i] - \mathrm{E}[f \mid \boldsymbol{X}_{i-1}, X_i = a_i']| \leq c_i$$

then

$$\Pr(|f - \mathrm{E}[f]| > t) \leq 2e^{-t^2/(2\sum_i c_i^2)}.$$

Lemma A.7 ([40], Corollary 5.2)**.** *Suppose that* $f(x_1, \ldots, x_n)$ *satisfies the Lipshitz property where* $|f(\boldsymbol{a}) - f(\boldsymbol{a}')| \leq c_i$ *whenever* $\boldsymbol{a}$ *and* $\boldsymbol{a}'$ *differ in just the* i*-th coordinate. If* $X_1, \ldots, X_n$

are independent random variables, then

$$|\mathrm{E}[f \mid \boldsymbol{X}_{i-1}, X_i = a_i] - \mathrm{E}[f \mid \boldsymbol{X}_{i-1}, X_i = a_i']| \leq c_i$$

Lemma A.8 ([40], Equation (8.5))**.** *Let $X_1, \ldots, X_n$ be an arbitrary set of random variables and let $f = f(X_1, \ldots, X_n)$ be such that $\mathrm{E}[f]$ is finite. For $1 \leq i \leq n$, suppose there exists σ_i^2 such that for any $\boldsymbol{X}_{i-1}$,*

$$\mathrm{Var}(\mathrm{E}[f \mid \boldsymbol{X}_i] - \mathrm{E}[f \mid \boldsymbol{X}_{i-1}] \mid \boldsymbol{X}_{i-1}) \leq \sigma_i^2$$

Also suppose that there exists M such that for $1 \leq i \leq n$, $|\mathrm{E}[f \mid \boldsymbol{X}_i] - \mathrm{E}[f \mid \boldsymbol{X}_{i-1}]| \leq M$. Then,

$$\Pr(f > \mathrm{E}[f] + t) \leq e^{-\frac{t^2}{2(\sum_{i=1}^n \sigma_i^2 + Mt/3)}}.$$

Appendix B

Publications Arising from this Dissertation

- A. Dornhaus, N. Lynch, T. Radeva, and H.-H. Su. Distributed task allocation in ant colonies. submitted.
- R. Duan, S. Pettie, and H.-H. Su. Scaling algorithms for weighted matching in general graphs. *CoRR*, abs/1411.1919, 2015.
- M. Elkin, S. Pettie, and H.-H. Su. (2Δ-1)-edge-coloring is much easier than maximal matching in the distributed setting. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 355–370, 2015.
- D. Nanongkai and H.-H. Su. Almost-tight distributed minimum cut algorithms. In *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 439–453. 2014.
- K.-M. Chung, S. Pettie, and H.-H. Su. Distributed algorithms for Lovász local lemma and graph coloring. In *Proc. 33rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 134–143, 2014.
- S. Pettie and H.-H. Su. Distributed coloring algorithms for triangle-free graphs. *Information and Computation*, 243(0):263 – 280, 2015. Preliminary version appeared in *40th Intl. Colloq. on Automata, Languages and Programming (ICALP)*, pages 681–693, 2013.
- R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1413–1424, 2012.

Bibliography

[1] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.

[2] N. Alon. A parallel algorithmic version of the local lemma. *Random Structures & Algorithms*, 2(4):367–378, 1991.

[3] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567 – 583, 1986.

[4] N. Alon, M. Krivelevich, and B. Sudakov. Coloring graphs with sparse neighborhoods. *Journal of Combinatorial Theory, Series B*, 77(1):73 – 82, 1999.

[5] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[6] M. L. Balinsky and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Sci.*, 10(3):578–593, 1964.

[7] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Computing*, 22:363–379, 2010.

[8] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *J. ACM*, 58(5):23, 2011.

[9] L. Barenboim and M. Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013.

[10] L. Barenboim, M. Elkin, and F. Kuhn. Distributed $(\Delta + 1)$-coloring in linear (in Δ) time. *SIAM J. Comput.*, 43(1):72–95, 2014.

[11] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. In *Proc. IEEE 53rd Symposium on Foundations of Computer Science (FOCS)*, pages 321 – 330, oct. 2012.

[12] J. Beck. An algorithmic approach to the Lovász local lemma. I. *Random Structures & Algorithms*, 2(4):343–365, 1991.

[13] G. Birkhoff. Tres observaciones sobre el elgebra lineal. *Universidad Nacional de Tucuman, Revista A*, 5(1–2):147–151, 1946.

[14] B. Bollobás. Chromatic number, girth and maximal degree. *Discrete Mathematics*, 24(3):311 – 314, 1978.

[15] E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 263(1376):1565–1569, 1996.

[16] O. V Borodin and A. V Kostochka. On an upper bound of a graph's chromatic number, depending on the graph's degree and density. *Journal of Combinatorial Theory, Series B*, 23(2-3):247–250, 1977.

[17] R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(02):194–197, 1941.

[18] P. A. Catlin. A bound on the chromatic number of a graph. *Discrete Math.*, 22(1):81 – 83, 1978.

[19] K. Chandrasekaran, N. Goyal, and B. Haeupler. Deterministic algorithms for the Lovász local lemma. *SIAM Journal on Computing*, 42(6):2132–2155, 2013.

[20] J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*.

[21] F. Chierichetti and A. Vattani. The local nature of list colorings for graphs of high girth. *SIAM J. Comput.*, 39(6):2232–2250, 2010.

[22] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[23] K.-M. Chung, S. Pettie, and H.-H. Su. Distributed algorithms for Lovász local lemma and graph coloring. In *Proc. 33rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 134–143, 2014.

[24] R. Cole and U. Vishkin. Approximate and exact parallel scheduling with applications to list, tree, and graph problems. In *Proc. FOCS'86*, pages 478–491, 1986.

[25] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[26] A. Cornejo, A. R. Dornhaus, N. A. Lynch, and R. Nagpal. Task allocation in ant colonies. In *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 46–60, 2014.

[27] W. H. Cunningham and A. B. Marsh, III. A primal algorithm for optimum matching. *Mathematical Programming Study*, 8:50–72, 1978.

[28] M. Cygan, H. N. Gabow, and P. Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter and matchings. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 531–540, 2012.

[29] A. Czumaj and C. Scheideler. A new algorithm approach to the general Lovász local lemma with applications to scheduling and satisfiability problems (extended abstract). In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 38–47, 2000.

[30] A. Czygrinow, M. Hanckowiak, and M. Karonski. Distributed $O(\Delta \log n)$-edge-coloring algorithm. In *ESA*, pages 345–355, 2001.

[31] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

[32] R. B. Dial. Algorithm 360: Shortest-path forest with topological ordering. *Commun. ACM*, 12(11):632–633, 1969.

[33] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[34] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. Math.*, 51(1):161–166, 1950.

[35] A. Dornhaus, N. Lynch, T. Radeva, and H.-H. Su. Distributed task allocation in ant colonies. submitted.

[36] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1, 2014.

[37] R. Duan, S. Pettie, and H.-H. Su. Scaling algorithms for weighted matching in general graphs. *CoRR*, abs/1411.1919, 2015.

[38] R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1413–1424, 2012.

[39] D. Dubhashi, D. A. Grable, and A. Panconesi. Near-optimal, distributed edge colouring via the nibble method. *Theor. Comput. Sci.*, 203(2):225–251, August 1998.

[40] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[41] J. Edmonds. Maximum matching and a polyhedron with $0,1$-vertices. *J. Res. Nat. Bur. Standards Sect. B*, 69B:125–130, 1965.

[42] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[43] J. Edmonds and E. L. Johnson. Matching, Euler tours, and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[44] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[45] M. Elkin. Personal communication.

[46] M Elkin. Distributed approximation: A survey. *SIGACT News*, 35(4):40–57, 2004.

[47] M. Elkin, S. Pettie, and H.-H. Su. (2Δ-1)-edge-coloring is much easier than maximal matching in the distributed setting. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 355–370, 2015.

[48] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hanjal, R. Rado, and V. T. Sós, editors, *Infinite and Finite Sets*, volume 11, pages 609–627. North-Holland, 1975.

[49] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, 1995.

[50] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[51] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

[52] H. N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(2):221–234, 1976.

[53] H. N. Gabow. Scaling algorithms for network problems. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–257, 1983.

[54] H. N. Gabow. A scaling algorithm for weighted matching on general graphs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 90–100, 1985.

[55] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443, 1990.

[56] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259 – 273, 1995.

[57] H. N. Gabow, Z. Galil, and T. H. Spencer. Efficient implementation of graph algorithms using contraction. *J. ACM*, 36(3):540–572, 1989.

[58] H. N. Gabow and P. Sankowski. Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In *Proceedings 54th Annual IEEE Symposium on Foundations of Computer Science FOCS*, pages 137–146, 2013. Full version available at arXiv:1304.6740.

[59] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985.

[60] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[61] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.

[62] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–14, 1962.

[63] Z. Galil, S. Micali, and H. N. Gabow. An $o(ev \log v)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM J. Comput.*, 15(1):120–130, 1986.

[64] J. A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

[65] M. Ghaffari and F. Kuhn. Distributed minimum cut approximation. In *Proc. 27th Symposium on Distributed Computing (DISC)*, pages 1–15. 2013.

[66] A. V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995.

[67] A. V. Goldberg and A. V. Karzanov. Maximum skew-symmetric flows and matchings. *Math. Program., Ser. A*, 100:537–568, 2004.

[68] A. V. Goldberg and R. Kennedy. Global price updates help. *SIAM J. Discrete Mathematics*, 10(4):551–572, 1997.

[69] A. V. Goldberg and S. A. Plotkin. Parallel (Δ+1)-coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241 – 245, 1987.

[70] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Discrete Math.*, 1(4):434–446, 1988.

[71] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[72] D. M. Gordon, B. C. Goodwin, and L. E. H. Trainor. A parallel distributed model of the behaviour of ant colonies. *Journal of theoretical Biology*, 156(3):293–307, 1992.

[73] D. A. Grable and A. Panconesi. Nearly optimal distributed edge coloring in $O(\log \log n)$ rounds. *Random Structures & Algorithms*, 10(3):385–405, 1997.

[74] D. A. Grable and A. Panconesi. Fast distributed algorithms for Brooks–Vizing colorings. *Journal of Algorithms*, 37(1):85 – 120, 2000.

[75] B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovász local lemma. *J. ACM*, 58(6):28, 2011.

[76] D. G. Harris. Lopsidependency in the Moser-Tardos framework: Beyond the lopsided Lovász local lemma. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1792–1808, 2015.

[77] D. G. Harris and A. Srinivasan. The Moser-Tardos framework with partial resampling. In *Proc. 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 469–478, 2013.

[78] D. G. Harris and A. Srinivasan. A constructive algorithm for the Lovász local lemma on permutations. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 907–925, 2014.

[79] N. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM J. Comput.*, 39(2):679–702, 2009.

[80] P. E. Haxell. A note on vertex list colouring. *Comb. Probab. Comput.*, 10(4):345–347, July 2001.

[81] H. Hind, M. Molloy, and B. Reed. Colouring a graph frugally. *Combinatorica*, 17(4):469–482, 1997.

[82] F. L. Hitchcock. The distribution of a product from several sources to numerous localities. *J. Math. Physics*, 20:224–230, 1941.

[83] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[84] C.-C. Huang and T. Kavitha. Efficient algorithms for maximum weight matchings in general graphs with small edge weights. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1400–1412, 2012.

[85] C. G. J. Jacobi. De investigando ordine systematis aequationum differentialum vulgarium cujuscunque. *Borchardt Journal für die reine und angewandte Mathematik*, LXIV(4):297–320, 1865. Reproduced in *C. G. J. Jacobi's gesammelte Werke, fünfter Band*, K. Weierstrass, Ed., Berlin, Bruck und Verlag von Georg Reimer, 1890, pp. 193–216.

[86] M. S. Jamall. A Brooks' Theorem for Triangle-Free Graphs. *ArXiv e-prints*, June 2011.

[87] M. S. Jamall. A Coloring Algorithm for Triangle-Free Graphs. *ArXiv e-prints*, January 2011.

[88] M. S. Jamall. *Coloring Triangle-Free Graphs and Network Games*. Dissertation, University of California, San Diego, 2011.

[89] T. R. Jensen and B. Toft. *Graph coloring problems*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1995.

[90] Ö. Johansson. Simple distributed $\Delta + 1$-coloring of graphs. *Information Processing Letters*, 70(5):229 – 232, 1999.

[91] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[92] L. Kantorovitch. On the translocation of masses. *Doklady Akad. Nauk SSSR*, 37:199–201, 1942. English translation in *Management Science* 5(1):1–4, 1958.

[93] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM J. Comput.*, 31(1):18–26, 2001.

[94] D. R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-out algorithm. In *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 21–30, 1993.

[95] D. R. Karger. Random sampling in cut, flow, and network design problems. In *Proce. 26th ACM Symposium on Theory of Computing (STOC)*, pages 648–657, 1994.

[96] D. R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 424–432, 1994.

[97] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.

[98] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. *J. ACM*, 42:321–329, 1995.

[99] D. R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, pages 757–765, 1993.

[100] A. V. Karzanov. On finding maximum flows in networks with special structure and some applications [in Russian]. In *Mathematical Issues of Production Control*, volume 5, pages 81–94. Moscow State University Press, Moscow, 1973. English translation available from the author's website.

[101] A. V. Karzanov. Efficient implementations of Edmonds' algorithms for finding matchings with maximum cardinality and maximum weight. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 306–327. Nauka, Moscow, 1976.

[102] K. Kawarabayashi and M. Thorup. Deterministic global minimum cut of a simple graph in near-linear time. *CoRR*, abs/1411.5123, 2014.

[103] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20(6):391–402, 2008.

[104] J. H. Kim. On brooks' theorem for sparse graphs. *Combinatorics, Probability and Computing*, 4:97–132, 1995.

[105] D. E. Knuth. *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[106] K. Kolipaka and M. Szegedy. Moser and Tardos meet Lovász. In *Proceedings 43rd ACM Symposium on Theory of Computing (STOC)*, pages 235–244, 2011.

[107] A. V. Kostochka and N. P. Mazuronva. An inequality in the theory of graph coloring. *Metody Diskret. Analiz.*, 30:23–29, 1977.

[108] K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in $\tilde{O}(\sqrt{\log n})$ bit rounds.

[109] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Lower and upper bounds for distributed packing and covering. Technical Report 443, ETH Zürich, 2004.

[110] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local computation: Lower and upper bounds. *CoRR*, abs/1011.5470, 2010.

[111] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 7–15, 2006.

[112] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[113] H. W. Kuhn. On combinatorial properties of matrices. *George Washington University Logistics Papers*, 11:1–11, 1955. English translation of J. Egerváry, Matrixok kombinatorius tulajdonságairól, *Matematikai és Fizikai Lapok 38*, 16–28, 1931.

[114] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[115] M. K. Kwan. Graphic programming using odd or even points. *Chinese Mathematics*, 1:273–277, 1962.

[116] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart & Winston, New York, 1976.

[117] J. Lawrence. Covering the vertex set of a graph with subgraphs of smaller degree. *Discrete Mathematics*, 21(1):61 – 68, 1978.

[118] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, February 1992.

[119] Z. Lotker, B. Patt-Shamir, and A. Rosén. Distributed approximate matching. *SIAM J. Comput.*, 39(2):445–460, 2009.

[120] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[121] D. W. Matula. A linear time $2 + \epsilon$ approximation algorithm for edge connectivity. In *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 500–504, 1993.

[122] S. Micali and V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

[123] A. Mądry. Navigating central path with electrical flows: From flows to matchings and back. In *Proceedings 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 253–262, 2013.

[124] M. Molloy and B. Reed. Further algorithmic aspects of the local lemma. In *Proc. 30th ACM Symposium on Theory of Computing (STOC)*, pages 524–529, 1998.

[125] M. Molloy and B. Reed. *Graph Colouring and the Probabilistic Method*. Algorithms and Combinatorics. Springer, 2001.

[126] M. Molloy and B. Reed. Asymptotically optimal frugal colouring. *J. Comb. Theory Ser. B*, 100(2):226–246, March 2010.

[127] R. A. Moser. Derandomizing the Lovász local lemma more effectively. *CoRR*, abs/0807.2120, 2008.

[128] R. A. Moser. A constructive proof of the Lovász local lemma. In *Proc. 41st ACM symposium on Theory of computing (STOC)*, pages 343–350, 2009.

[129] R. A. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57(2):11, 2010.

[130] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proc. 45th Symp. on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

[131] J. Munkres. Algorithms for the assignment and transportation problems. *J. Soc. Indust. Appl. Math.*, 5:32–38, 1957.

[132] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discret. Math.*, 5(1):54–66, 1992.

[133] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *STOC*, pages 565–573, 2014.

[134] D. Nanongkai and H.-H. Su. Almost-tight distributed minimum cut algorithms. In *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 439–453. 2014.

[135] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. s1-36(1):445–450, 1961.

[136] S. Navlakha and Z. Bar-Joseph. Distributed information processing in biological and computational systems. *Communications of ACM*, 58(1):94–102, 2015.

[137] F. Ollivier. Looking for the order of a system of arbitrary ordinary differential equations. *Appl. Algebra Eng. Commun. Comput.*, 20(1):7–32, 2009. English translation of: C. G. J. Jacobi, De investigando ordine systematis aequationum differentialum vulgarium cujuscunque," *Borchardt Journal für die reine und angewandte Mathematik* 65(4), 1865, pp. 297–320, also reproduced in: C. G. J. Jacobi, *Gesammelte Werke, Vol. 5, K. Weierstrass, Ed., Berlin, Bruck und Verlag von Georg Reimer, 1890, pp. 193–216.*,.

[138] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Math. Program.*, 54:41–56, 1992.

[139] S. W. Pacala, D. M. Gordon, and H. C. J. Godfray. Effects of social group size on information transfer and task allocation. *Evol Ecol*, 10(2):127–165, mar 1996.

[140] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.

[141] A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356 – 374, 1996.

[142] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.

[143] W. Pegden. An extension of the Moser-Tardos algorithmic local lemma. *CoRR*, abs/1102.2853, 2011.

[144] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[145] S. Pemmaraju and A. Srinivasan. The randomized coloring procedure with symmetry-breaking. In *Proc. 35th Int'l Colloq. on Automata, Languages, and Programming (ICALP)*, volume 5125 of *LNCS*, pages 306–319. 2008.

[146] H. M. Pereira and D. M. Gordon. A trade-off in task allocation between sensitivity to the environment and response time. *Journal of theoretical biology*, 208(2):165–184, 2001.

[147] S. Pettie. Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time. In *Proceedings 16th Int'l Symposium on Algorithms and Computation (ISAAC)*, pages 964–973, 2005. Full version available at arXiv:1407.1910.

[148] S. Pettie. A simple reduction from maximum weight matching to maximum cardinality matching. *Inf. Process. Lett.*, 112(23):893–898, 2012.

[149] S. Pettie and H.-H. Su. Distributed coloring algorithms for triangle-free graphs. *Information and Computation*, 243(0):263 – 280, 2015. Preliminary version appeared in *40th Intl. Colloq. on Automata, Languages and Programming (ICALP)*, pages 681–693, 2013.

[150] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proc. 32nd IEEE Foundations of Computer Science (FOCS)*, pages 495–504, 1991.

[151] D. Pritchard and R. Thurimella. Fast computation of small cuts via cycle space sampling. *ACM Transactions on Algorithms*, 7(4):46, 2011.

[152] B. Reed. The list colouring constants. *Journal of Graph Theory*, 31(2):149–153, 1999.

[153] B. Reed and B. Sudakov. Asymptotically the list colouring constants are 1. *Journal of Combinatorial Theory, Series B*, 86(1):27 – 37, 2002.

[154] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proceedings of the 2nd Symposium on Innovations in Computer Science(ICS)*, pages 223–238, 2011.

[155] P. Sankowski. Weighted bipartite matching in matrix multiplication time. In *Proceedings 33rd Int'l Symposium on Automata, Languages, and Programming (ICALP)*, pages 274–285, 2006.

[156] J. Schneider and R. Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 35–44, 2008.

[157] J. Schneider and R. Wattenhofer. A new technique for distributed symmetry breaking. In *Proc. 29th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 257–266, 2010.

[158] J. Spencer. Asymptotic lower bounds for Ramsey functions. *Discrete Mathematics*, 20:69–76, 1977.

[159] A. Srinivasan. Improved algorithmic versions of the Lovász local lemma. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 611–620, 2008.

[160] M. Stoer and F. Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997.

[161] M. Szegedy and S. Vishwanathan. Locality based graph coloring. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, pages 201–207, 1993.

[162] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999.

[163] M. Thorup. Equivalence between priority queues and sorting. In *Proc. 43rd Symp. on Foundations of Computer Science (FOCS)*, pages 125–134, 2002.

[164] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024 (electronic), 2004.

[165] M. Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.

[166] R. Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *Journal of Algorithms*, 23(1):160 – 179, 1997.

[167] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.

[168] W. T. Tutte. On the problem of decomposing a graph into n connected factors. s1-36(1):221–230, 1961.

[169] V. V. Vazirani. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR*, abs/1210.4594, 2012.

[170] V. G. Vizing. Some unsolved problems in graph theory. *Uspekhi Mat. Nauk*, 23(6(144)):117 – 134, 1968.

[171] J. von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume II, pages 5–12. Princeton University Press, 1953.

[172] V. H. Vu. A general upper bound on the list chromatic number of locally sparse graphs. *Comb. Probab. Comput.*, 11(1):103–111, January 2002.

[173] N. E. Young. Randomized rounding without solving the linear program. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 170–178, 1995. full version available at: `http://arxiv.org/abs/cs.DS/0205036`.