Fast Algorithms for Edge Coloring Graphs

Martín Costa

Thesis submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Department of Computer Science
University of Warwick

December 2025

Para mis padres,

Pablo y Verónica,

y mis abuelos,

Oscar, María, Ricardo y Lucía

Contents

List of Figures

Acknowledgments

First and foremost, I would like to thank my supervisor, *Sayan Bhattacharya*. I started working with Sayan at the end of my second year of undergrad, where we immediately dove into tackling challenging open problems in algorithms. Sayan consistently encouraged me to pursue my own research ideas while simultaneously being there to provide support and guidance every step of the way. The frequent, informal, and often impromptu meetings we had on an almost daily basis during my initial years fostered an amazing research environment, enabling me to quickly gain independence and develop as a researcher.

I would like to thank all of my coauthors: *Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Din Carmon, Ermiya Farokhnejad, Hendrik Fichtenberger, Naveen Garg, Shaofeng Jiang, Yaonan Jin, Jakub Łącki, Silvio Lattanzi, André Linhares, Jianing Lou, Nadav Panski, Nikos Parotsidis, Shay Solomon, and Tianyi Zhang*. Working with you has made research an amazing experience.

I would also like to thank... *Shay Solomon*, for all of the guidance over the years. *Nikos Parotsidis*, for hosting me as a student researcher at Google. *Sepehr Assadi*, for hosting me at the University of Waterloo. *Gramoz Goranci* and *Peter Kiss*, for hosting me at the University of Vienna, and for the many chats about research and life over the years. *Maria Ferreiro*, for the extensive assistance with all non-scientific matters.

Outside of the academic world, I would like to thank... *Henry Sinclair-Banks*, *Ninad Rajgopal*, *Ian Mertz*, *Gozde Gunesli*, and *Ermiya Farokhnejad*, for making Coventry more entertaining. *Harry Roberts*, *Harry Thompson*, and *Joe Lang*, for the many beers and chats over the past few years. *Alexander Aktas* and *James Gulliford*, for being hilarious individuals. My teachers *Owen Edwards*, *Gareth Johnson*, *Stuart Watson*, and *Paul Williams*, for helping me start my journey in mathematics, as well as my many friends from school and university who have been a part of this process.

Finally, I would like to thank my family, *Pablo*, *Verónica*, and *Facundo*, for their endless support.

Declaration

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree.

The work presented was carried out by the author; it is fundamental research in nature and does not include data or data analytics. Much of the work presented in this thesis has been published in peer-reviewed conferences; substantial work was carried out by the author throughout all phases, including conceptualization, formal analysis, and writing. This thesis is based on results contained in the following publications.

List of Works Included in the Thesis:

- [ABB+25] Vizing's Theorem in Near-Linear Time
  with Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Shay Solomon, and Tianyi Zhang
  Symposium on Theory of Computing (**STOC**) 2025 (Chapter 7)
  Received the Best Paper Award at STOC'25

- [ABB+26] Vizing's Theorem in Deterministic Almost-Linear Time
  with Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Shay Solomon, and Tianyi Zhang
  Symposium on Discrete Algorithms (**SODA**) 2026 (Chapter 8)

- [BCSZ25] Even Faster $(\Delta + 1)$-Edge Coloring via Shorter Multi-Step Vizing Chains
  with Sayan Bhattacharya, Shay Solomon, Tianyi Zhang
  Symposium on Discrete Algorithms (**SODA**) 2025 (Chapter 9)

- [BCPS24c] Nibbling at Long Cycles: Dynamic (and Static) Edge Coloring in Optimal Time
  with Sayan Bhattacharya, Nadav Panski, Shay Solomon
  Symposium on Discrete Algorithms (**SODA**) 2024 (Chapter 10)

List of Works Not Included in the Thesis:

- [BCF+25b] Almost Optimal Fully Dynamic k-Center Clustering with Recourse
  with Sayan Bhattacharya, Ermiya Farokhnejad, Silvio Lattanzi, Nikos Parotsidis
  International Conference on Machine Learning (**ICML**) 2025

- [CF25] Deterministic k-Median Clustering in Near-Optimal Time
  with Ermiya Farokhnejad
  International Colloquium on Automata, Languages, and Programming (**ICALP**) 2025

- [BCF25a] Fully Dynamic k-Median with Near-Optimal Update Time and Recourse
  with Sayan Bhattacharya, Ermiya Farokhnejad
  Symposium on Theory of Computing (**STOC**) 2025

- [BCG+24] Fully Dynamic k-Clustering with Fast Update Time and Small Recourse
  with Sayan Bhattacharya, Naveen Garg, Silvio Lattanzi, Nikos Parotsidis
  Symposium on Foundations of Computer Science (**FOCS**) 2024

- [BCC+24] Faster $(\Delta + 1)$-Edge Coloring: Breaking the $m\sqrt{n}$ Time Barrier
  with Sayan Bhattacharya, Din Carmon, Shay Solomon, Tianyi Zhang
  Symposium on Foundations of Computer Science (**FOCS**) 2024

- [BCPS24b] Density-Sensitive Algorithms for $(\Delta + 1)$-Edge Coloring
  with Sayan Bhattacharya, Nadav Panski, Shay Solomon
  European Symposium on Algorithms (**ESA**) 2024

- [BCPS24a] Arboricity-Dependent Algorithms for Edge Coloring
  with Sayan Bhattacharya, Nadav Panski, Shay Solomon
  Scandinavian Symposium on Algorithm Theory (**SWAT**) 2024

- [BCLP23] Fully Dynamic k-Clustering in $\tilde{O}(k)$ Update Time
  with Sayan Bhattacharya, Silvio Lattanzi, Nikos Parotsidis
  Neural Information Processing Systems (**NeurIPS**) 2023

This thesis does not address many of the results that I have published throughout my PhD. The papers [BCPS24a, BCPS24b, BCC+24] are omitted either due to being subsumed by subsequent works or due to space constraints, and papers [BCLP23, BCG+24, BCF25a, CF25, BCF+25b] are omitted since they are based on clustering algorithms and unrelated to the topic of this thesis.

Abstract

Designing near-optimal algorithms for fundamental graph problems is a central research agenda within the field of algorithms, with a significant body of work being devoted to this task since the inception of computer science. While modern computational paradigms have led to the rise of various different computational models, the complexity of many basic problems is still far from being fully understood in the classic static sequential setting. In recent years, landmark breakthroughs in continuous optimization have yielded near-optimal algorithms for maximum flow and bipartite matching, sparking a quest for new equally efficient combinatorial solutions.

Edge coloring is a fundamental problem in graph theory and has been studied extensively by both mathematicians and computer scientists for many decades. This thesis tackles the challenge of designing near-optimal algorithms for edge coloring in the static and dynamic settings. As our main result, we introduce a simple, combinatorial algorithm for computing a near-optimal edge coloring of a graph in near-linear time, resolving a major, long-standing open problem by settling the complexity of edge coloring in the static sequential model. Furthermore, we design new dynamic algorithms for approximately edge coloring graphs, including an edge coloring algorithm with constant update time, which also leads to the first exact linear time algorithm for this problem.

Part I

Extended Synopsis

Chapter 1

Introduction

An edge coloring of a graph is an assignment of labels, referred to as *colors*, to the edges of the graph, such that no two edges sharing an endpoint receive the same color. More formally, given a simple undirected graph $G = (V, E)$ on n vertices and m edges, as well as a positive integer μ, a *(proper) μ-edge coloring* (shortly, μ-coloring) $\chi : E \longrightarrow \{1, \ldots, \mu\}$ of G assigns a color $\chi(e)$ to each edge $e \in E$ such that any two adjacent edges $e, f \in E$ receive distinct colors $\chi(e) \neq \chi(f)$. In other words, a μ-coloring is a partition of the edges into μ different *matchings*.

Edge colorings of graphs are fundamental objects in graph theory [Viz64, Sha49, Gal95], and are actively studied within combinatorics. Initially motivated by applications in scheduling [Got62, LL78], edge coloring and other related problems, such as maximum matching, have been studied extensively by computer scientists due to their simplicity and broad applicability.

We can always trivially compute an edge coloring of G by simply assigning each edge a unique color. Thus, the problem of computing an edge coloring is only interesting when we attempt to minimize the number of colors used. The minimum number of colors required to edge color a graph G is referred to as the *chromatic index* of G, and is denoted by $\chi'(G)$. If G has maximum vertex degree Δ, then any edge coloring of G requires at least Δ different colors, since each of the edges incident on some maximum degree vertex must be assigned distinct colors. Remarkably, a classical theorem by Vizing shows that $\Delta + 1$ colors are always sufficient [Viz64]. Taken together, this gives very tight upper and lower bounds on the chromatic index of G, since $\Delta \leq \chi'(G) \leq \Delta + 1$.

Designing near-optimal algorithms for fundamental graph problems is an important research agenda within the field of algorithms. Algorithmic breakthroughs for edge coloring were first achieved in the classic static sequential model of computation. The proof of Vizing's theorem is inherently algorithmic and immediately leads to an $O(mn)$ time algorithm for computing a $(\Delta+1)$-edge coloring, giving the first polynomial time algorithm for this basic task. On the other hand, it was proven by [Hol81] that it is NP-hard to distinguish whether the chromatic index of a graph is Δ or $\Delta + 1$, and therefore $\Delta + 1$ is the best bound we can hope for with polynomial time algorithms.

As a consequence of new technologies emerging and widening domains of application, computational paradigms have shifted over time. Over the past few decades, large-scale distributed systems and the need to handle massive, rapidly changing datasets have become increasingly com-

mon, driving a shift away from the traditional static sequential setting. This evolution requires us to develop new models and algorithms, as we must now operate under fundamentally different constraints and objectives, such as minimizing communication or handling dynamic data. Consequently, there is now a vibrant body of work on designing edge coloring algorithms across a wide range of computational models, such as the *dynamic* [BM17, BCHN18, DHZ19, Chr23], *distributed* [DGP98, CHL+20, Ber22, BBKO22, JMS25], *parallel* [KS87, LSH96], *online* [BMN92, CPW19, KLS+22, BSVW24, BSVW25a, DGS25, BSVW25b], and *streaming* [SB24, CCZ25] settings. Understanding the complexity of edge coloring and related problems within these computational models has played a significant role in developing our understanding of these different settings.

Despite the significant progress that has been made in developing efficient algorithms for edge coloring, we are still far from understanding the complexity of this fundamental problem. This leads us to the following question, which is the main focus of this thesis:

Question 1. *Can we design near-optimal algorithms for edge coloring?*

Understanding the Fundamentals: Static Sequential Algorithms. While modern computational paradigms have led to the rise of various different computational models, the complexity of many basic problems is still far from being fully understood in the classic static sequential model of computation. Building on Vizing's original $(\Delta+1)$-coloring algorithm [Viz64], subsequent improvements led to a running time of $\tilde{O}(m\sqrt{n})$, where the state-of-the-art running time for this problem got stuck for over 40 years [Arj82, GNK+85]. This running time of $\tilde{O}(m\sqrt{n})$ also coincides with classical barriers for other basic graph problems, such as *maximum matching* [MV80, DPS18], or *negative-weight shortest paths* [GT89, Gol95]. The similarity with the barrier for maximum matching is particularly striking—these two problems are intimately related in many different models, with insights for one problem often finding applications for the other [CPW19, GG24].

However, recent lines of work have made significant progress in breaking through similar barriers for other related graph problems: breakthroughs in continuous optimization led to a remarkably fast *almost-linear* $m^{1+o(1)}$ time algorithm for maximum flow, a key generalization of bipartite matching [CKL+25]. This success sparked a new quest for equally efficient *combinatorial* algorithms for these problems, which do not rely on continuous optimization or heavy dynamic data structures. This line of work resulted in another landmark achievement: a $\tilde{O}(n^2)$ time algorithm for maximum flow, which is *near-linear* for dense graphs [CK24, BBST24, BBL+25].

Given these recent advancements, we ask the following ambitious question, which aims to settle the complexity of this fundamental problem in the classic static sequential setting.

Question 2. *Can we design an algorithm for $(\Delta+1)$-coloring that runs in near-linear time?*

Dealing with Massive and Evolving Data: Dynamic Algorithms. In many modern applications, the underlying data is not static but changes over time. Communication networks, social

networks, and online platforms are inherently dynamic. Processing these massive, evolving datasets from scratch after every change is computationally infeasible. This has spurred the development of *dynamic algorithms*, which can maintain solutions as the underlying data undergoes continuous updates. For dynamic graph algorithms, these updates usually correspond to edge insertions and deletions from the graph. The primary goal is to process each update significantly faster than recomputing the solution from scratch, while also ensuring that we maintain a good solution. Furthermore, designing efficient dynamic algorithms can be viewed as a natural extension of developing efficient static algorithms: a dynamic algorithm immediately gives a static algorithm by simply feeding it the entire graph one edge at a time. The ultimate goal is to design a dynamic algorithm that matches the state-of-the-art for the static setting.

Dynamic edge coloring has received a lot of attention in recent years, with a significant body of work devoted to designing efficient algorithms for maintaining edge colorings of dynamic graphs [BM17, BCHN18, DHZ19, BGW21, Chr23, BCPS24a]. This has led to the discovery of various new techniques—most notably the development of *multi-step Vizing chains* [DHZ19], which have found many applications across static, dynamic and distributed settings [Ber22, Chr23, Dha24]. Using these techniques, [Chr23] showed how to maintain a $(1+\epsilon)\Delta$-coloring with $\text{poly}(1/\epsilon, \log n)$ update time, for any $\epsilon > 0$. While there are some barriers to improving upon this update time [CHL+20], it is not clear whether or not this tradeoff is the best we can hope to achieve. This leads us to the second question that we address in this thesis, which aims to design near-optimal dynamic algorithms for edge coloring.

Question 3. *Can we design near-optimal dynamic algorithms for edge coloring?*

1.1 Our Contributions

We now summarize the main contributions of this thesis, which introduce various new techniques and algorithms for edge coloring in the static and dynamic settings, leading to significant progress in our understanding of this fundamental computational problem.

Settling the Complexity of Vizing's Theorem

A foundational challenge in this area is the efficient implementation of Vizing's theorem [Viz64], with the ultimate goal being to resolve Question 2. As highlighted above, the first steps towards designing more efficient algorithms for this problem were taken by [Arj82] and [GNK+85], who showed how to get a running time to $O(m\sqrt{n \log n})$, which was later improved to $O(m\sqrt{n})$ with a sharper analysis [Sin19]. The running time for this task got stuck at $\tilde{O}(m\sqrt{n})$, and *there was no polynomial improvement on the running time of this fundamental problem for over 40 years.*

Classical algorithms for this task, building on Vizing's techniques, construct a coloring by 'extending' it to one edge at a time. If no color is immediately available for an uncolored edge, the algorithm must recolor existing edges. This is typically done by flipping colors along a 'Vizing

chain'—a path of edges alternating between two colors. The length of this chain is proportional to the time taken to extend the coloring, making long chains computationally expensive to handle. The specific pair of colors in this chain defines the uncolored edge's 'color type'. For decades, the primary bottleneck for these algorithms was the need to manage up to $O(\Delta^2)$ distinct color types, since the average length of the corresponding Vizing chains can be infeasibly large.

In this thesis, we introduce a new approach that shatters this long-standing paradigm. Instead of the traditional one-by-one approach, we introduce a randomized subroutine that colors edges in large batches. The core of this new algorithm is a novel 'type sparsification' technique, which collapses the number of distinct Vizing chain types from a quadratic $O(\Delta^2)$ to a linear $O(\Delta)$, allowing us to find much shorter Vizing chains and leading to vastly better amortized efficiency for extending the coloring to edges. Using this technique, we developed a randomized algorithm that computes a $(\Delta+1)$-edge coloring in near-linear $O(m \log \Delta)$ time. This result resolves the time complexity of this problem in the static sequential setting up to a logarithmic factor, providing a near-optimal algorithm. Furthermore, this matches the fastest known running time for computing a Δ-coloring in bipartite graphs, where a very simple $O(m \log \Delta)$ time algorithm is known [COS01].

Theorem 1.1.1. *There is an algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $O(m \log \Delta)$ time with high probability.*

Preceding Work: Towards a Near-Linear Time Algorithm

Our earlier work [BCC+24] achieved the first polynomial improvement over the long-standing $\tilde{O}(m\sqrt{n})$ time barrier by tackling another core challenge of coloring general graphs: the complex structure of 'Vizing fans' (a complicated gadget required to construct Vizing chains), leading to an $\tilde{O}(mn^{1/3})$ time algorithm. We introduced a new algorithmic framework based on novel structural observations on how different Vizing fans interact with each other. At a high level, we showed that it is possible to 'untangle' different Vizing fans that intersect each other, ensuring they correspond to distinct Vizing chains. This deeper understanding allowed us to design an algorithm that finds significantly shorter Vizing chains on average, breaking through this historical bottleneck. Independently and concurrently, [Ass25] showed how to obtain a $\tilde{O}(n^2)$ time algorithm, which is also faster than $\tilde{O}(m\sqrt{n})$ for sufficiently dense graphs. This algorithm uses completely disjoint techniques and is based on adapting random walk-based sublinear algorithms for finding matchings in regular bipartite graphs to general graphs [GKK10]. Building on this, our subsequent work further refined these techniques to achieve another significant speedup [BCSZ25]. This follow-up paper introduces a new, highly efficient 'color extension subroutine' designed to color a single remaining edge. The key innovation is a new construction that produces substantially shorter multi-step Vizing chains—a powerful generalization of traditional Vizing chains [DHZ19, Chr23, Ber22]. However, we observed that these approaches were *fundamentally constrained by the need to extend the coloring one edge at a time*, which motivated our shift to the batch-processing paradigm that achieved a near-linear time algorithm. *The algorithms presented in this thesis build directly on the classical works of [Viz64, GNK+85] and do not rely on any of these recent developments.*

Deterministic Algorithms for Vizing's Theorem

Understanding the *deterministic* complexity of fundamental graph problems is also an important research direction within algorithms [vdBCP+23, HLRW24]. While our near-linear time algorithm for $(\Delta+1)$-coloring effectively resolves the complexity of this problem, this algorithm crucially uses randomization. In fact, at the heart of *all* of the recent works that have broken through the $\tilde{O}(m\sqrt{n})$ time barrier, there is some form of a randomized *sublinear* time algorithm. For example, our near-linear time algorithm repeatedly applies an $O(m/\Delta)$ time subroutine throughout sequential 'phases'; during each of these phases, the algorithm extends the coloring to an $\Omega(1/\Delta)$ proportion of the remaining uncolored edges, and thus completely colors the graph after $\tilde{O}(\Delta)$ phases. However, while sublinear time algorithms are a very powerful tool for designing randomized algorithms, they are almost always impossible to derandomize.

By generalizing the algorithmic framework that we use to obtain Theorem 1.1.1, we are able to entirely forego sublinear time algorithms. The key idea is to replace the randomized sublinear time subroutine with a deterministic almost-linear time algorithm that is capable of coloring much larger batches of edges in one go, leading to a *deterministic* $(\Delta+1)$-coloring algorithm that runs in almost-linear time, giving the first polynomial improvement over the $\tilde{O}(m\sqrt{n})$ time deterministic algorithm of [GNK+85].

Theorem 1.1.2. *There is a deterministic algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $m \cdot 2^{O(\sqrt{\log \Delta})} \cdot \log n = m^{1+o(1)}$ time.*

Algorithms for Edge Coloring Multigraphs

A generalization of Vizing's theorem for (loop-less) multigraphs shows that any multigraph G with maximum degree Δ and maximum edge multiplicity μ can be $(\Delta+\mu)$-edge colored [Viz65]. With some small modifications, the algorithm from Theorem 1.1.1 can be extend to Vizing's theorem for multigraphs.

Theorem 1.1.3. *There is an algorithm that, given a multigraph G with maximum degree Δ and maximum multiplicity μ, computes a $(\Delta+\mu)$-coloring of G in $O(m \log \Delta)$ time with high probability.*

With the same changes, Theorem 1.1.2 can also be extended to multigraphs.

Theorem 1.1.4. *There is a deterministic algorithm that, given a multigraph G with maximum degree Δ and maximum multiplicity μ, computes a $(\Delta+\mu)$-coloring of G in $m \cdot 2^{O(\sqrt{\log \Delta})} \cdot \log n = m^{1+o(1)}$ time.*

Another classical result in edge coloring is *Shannon's theorem*, which shows that any multigraph G of maximum degree Δ can be $\lfloor 3\Delta/2 \rfloor$-colored [Sha49]. Unlike Vizing's theorem for multigraphs, this bound does not depend on the maximum multiplicity of the graph. Shannon's original proof immediately leads to an $O(mn)$ time algorithm for computing such a coloring. Much more recently, [Dha24] gave an $O(n\Delta^{18})$ time algorithm for this task, giving the first improvement over Shannon's algorithm for very small Δ.

By leveraging a simple reduction from Shannon's theorem to Vizing's theorem, we obtain a near-linear time algorithm for computing such a coloring, leading to a near-optimal algorithm for this basic task.

Theorem 1.1.5. *There is an algorithm that, given a multigraph G with maximum degree Δ, computes a $\lfloor 3\Delta/2 \rfloor$-coloring of G in $O(m \log \Delta)$ time with high probability.*

Dynamic Algorithms via Shorter Multi-Step Vizing Chains

In the dynamic setting, we are given a graph G that evolves over time via a sequence of *updates* consisting of edge insertions and deletions, and our objective is to explicitly maintain an edge coloring of the graph during this process. Additionally, we assume that the maximum degree of G remains at most Δ throughout the sequence of updates, where Δ is a known parameter.

We can maintain a $(2\Delta-1)$-coloring in $O(\log \Delta)$ update time [BM17, BCHN18], which essentially requires dynamizing the greedy algorithm using a variant of binary search. If we wish to move beyond the greedy threshold of $2\Delta - 1$, then there are two further results. We know how to maintain a $(1+\epsilon)\Delta$-coloring in $O(\log^7 n/\epsilon^2)$ update time when $\Delta = \Omega(\log^2 n/\epsilon^2)$ [DHZ19], and a $(1+\epsilon)\Delta$-coloring in $O(\log^9 n \log^6 \Delta/\epsilon^6)$ update time with no restrictions on Δ [Chr23]. Both of these algorithms are based on multi-step Vizing chain constructions, which are a concatenation of multiple (carefully chosen) Vizing chains: given an uncolored edge e, we can construct a multi-step Vizing chain that can be used to extend the coloring to the edge e. Vizing's original algorithm shows that constructing a Vizing chain to extend the coloring to an edge takes $O(n)$ time. Multi-step Vizing chains are a direct generalization of this procedure, and can be much faster when Δ is sufficiently small or when we are allowed to have more than $\Delta + 1$ colors.

By building on the original multi-step Vizing chain algorithm of [DHZ19], we provide a new construction that is capable of producing much shorter multi-step Vizing chains for $(\Delta+1)$-coloring for a large range of values of Δ. We show that, given a partial $(\Delta+1)$-coloring with an uncolored edge, we can construct a $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$ length multi-step Vizing chain at this edge, allowing us to extend the coloring to this edge in time proportional to the length of the chain. The best previous time bound of any color extension subroutine for $(\Delta+1)$-coloring is either the trivial $O(n)$ by [Viz64] or the bound $\tilde{O}(\Delta^6)$ by Bernshteyn [Ber22], which also uses multi-step Vizing chains.

Theorem 1.1.6. *There is an algorithm that, given a dynamic graph G with maximum degree at most Δ, maintains a $(\Delta+1)$-coloring of G with $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$ worst-case update time, against an adaptive adversary, with high probability.*

Remark 1.1.7. *We note that the motivation behind designing the heavily optimized dynamic algorithm in Theorem 1.1.6 was to produce faster static algorithms for $(\Delta+1)$-coloring. In particular, combining this dynamic algorithm with previous works [BCC+24, Ass25] led to polynomial improvements over the state-of-the-art running time for computing a $(\Delta+1)$-coloring [BCSZ25]. However, these results have all been subsumed by Theorem 1.1.1, and hence are omitted from this thesis.*

Dynamic (and Static) Approximate Edge Coloring in Optimal Time

If we consider the task of dynamically maintaining a $(1+\epsilon)\Delta$-coloring for the regime where $\epsilon > 0$ is a small *constant*, can we improve upon the algorithms of [DHZ19, Chr23]? Given that both of these two results incur a *large* polylogarithmic factor in their update times, it is natural to ask the following question: can we bring this update time down all the way to $O(1)$?

A barrier to removing the polylogarithmic dependence in the update time is the lower bound of [CHL+20], which shows that, for any $1/\Delta \leq \epsilon \leq 1/3$, there exists a graph G and a $(1+\epsilon)\Delta$-coloring χ of G with an uncolored edge e such that extending χ to e requires changing the colors of $\Omega(\log(\epsilon n)/\epsilon)$ edges. This implies that any analysis that does not consider the structure of the edge coloring maintained by a dynamic algorithm will *not* be sufficient to get constant update time. Most dynamic edge coloring algorithms—such as the algorithms based on Vizing chains—are analyzed in this 'memoryless' manner, and assume that the edge coloring of the graph is completely arbitrary at the start of each update [DHZ19, Chr23].

By using different techniques, we were able to bypass this lower bound, developing an algorithm that achieves $\text{poly}(1/\epsilon)$ update time for maintaining a $(1+\epsilon)\Delta$-coloring, for constant $\epsilon > 0$, getting an optimal running time. At a high level, our algorithm works by maintaining an edge coloring from a 'stable' distribution defined by the output of a suitable static algorithm. The algorithm combines a new variant of the 'Nibble' method [BGW21] with a subsampling technique that was originally used in the online setting [KLS+24]. This subsampling allows us to split the input graph into subgraphs that are 'locally treelike', meaning that they have very few short cycles. These graphs are much easier to handle, leading to a simple analysis of an algorithm that is notoriously complicated to analyze [DGP98].

In the edge coloring literature, it is common to assume that $\Delta \geq \Omega(\text{polylog}(n))$ with certain techniques [BGW21, DHZ19, KLS+22]. In order for the concentration bounds in our analysis to go through, we require that $\Delta \geq \Delta^\star := (\log n/\epsilon)^{\Theta(\log(1/\epsilon)/\epsilon)}$.

Theorem 1.1.8. *There is an algorithm that, given a dynamic graph G with maximum degree at most $\Delta \geq (\log n/\epsilon)^{\text{poly}(1/\epsilon)}$, maintains a $(1+\epsilon)\Delta$-coloring of G with $\text{poly}(1/\epsilon)$ expected worst-case update time, against an oblivious adversary.*

As a corollary of our constant update time dynamic algorithm, we also obtain the first truly linear time static algorithm for $(1+\epsilon)\Delta$-coloring. Previously, it was not even known how to compute a greedy $(2\Delta-1)$-coloring in linear time.

Corollary 1.1.9. *There is an algorithm that, given a graph G with maximum degree at most $\Delta \geq (\log n/\epsilon)^{\text{poly}(1/\epsilon)}$, computes a $(1+\epsilon)\Delta$-coloring of G in $O(m\,\text{poly}(1/\epsilon))$ time with high probability.*

1.2 Organization

The rest of Part I (the extended abstract) is mainly devoted to providing a detailed technical overview of our algorithms and results. In Chapter 2, we briefly summarize the relevant history of

algorithms and techniques for static and dynamic edge coloring, setting the stage for the results of this thesis. In Chapter 3, we provide the basic technical background for this thesis. In Chapter 4, we give an overview of our static algorithms for $(\Delta+1)$-coloring. In Chapter 5, we give an overview of our dynamic algorithms. Finally, in Chapter 6, we discuss open questions and future research directions to expand on the work of this thesis. Subsequently, in Parts II and III, we present our complete algorithms and results for the static and dynamic settings, respectively.

Chapter 2

The Algorithmic History of Vizing's Theorem

In this chapter, we describe the historical development of algorithms for static and dynamic edge coloring, setting the stage for the results of this thesis. We begin by describing the proof of Vizing's theorem and overviewing the algorithms and techniques that have been used to design efficient $(\Delta + 1)$-coloring algorithms, as well as their limitations, followed by an overview of the algorithmic techniques used in the dynamic setting.

2.1 Vizing's Theorem

Vizing's theorem [Viz64] is a classical textbook result within algorithmic graph theory and combinatorics [BM76]. The following theorem summarizes the algorithm that is implicitly described in Vizing's original proof.

Theorem 2.1.1. *There is an algorithm that, given any graph G on n vertices and m edges with maximum degree Δ, computes a $(\Delta + 1)$-coloring of G in $O(mn)$ time.*[1]

2.1.1 Vizing's Theorem for Bipartite Graphs

We first explain a simplified variant of Vizing's algorithm for Δ-coloring *bipartite graphs*, which conveys the main ideas of the proof while significantly simplifying the technical details.

Let $G = (V, E)$ be a bipartite graph of maximum degree Δ. Vizing's algorithm starts by creating a *partial* coloring $\chi : E \longrightarrow [\Delta] \cup \{\perp\}$, where $\chi(e) = \perp$ for each $e \in E$, which denotes each edge being *uncolored*. The algorithm then scans through each edge $e \in E$ in an arbitrary order, and *extends* the coloring χ to the edge e, meaning that it modifies the coloring χ by changing the colors assigned to the already colored edges and also assigns a color to e. After extending the coloring to each edge, no edges remain uncolored, and χ is a Δ-coloring of G.

[1] Unless stated otherwise, we always assume graphs are simple and undirected.

This high-level strategy of extending the coloring to one edge at a time forms the basic framework that most static $(\Delta + 1)$-coloring algorithms follow; the interesting part lies in the procedure used to extend the coloring to an edge.

Extending the Coloring χ. Let $e = (u, v)$ be an edge that is currently uncolored, i.e. $\chi(e) = \perp$. In order to extend the coloring χ to e, we begin by identifying *missing colors* at the endpoints of e. We say that a color $c \in [\Delta]$ is missing at a vertex w if no edge incident on w has color c. In other words, the set of missing colors at w is defined as $\mathsf{miss}_\chi(w) := [\Delta] \setminus \{\chi(f) \mid f \in E, f \ni w\}$. Since u has maximum degree Δ, and the edge e is uncolored, there are at most $\Delta - 1$ colors assigned to edges incident on u. Thus, we have that $\mathsf{miss}_\chi(u) \neq \emptyset$, and $\mathsf{miss}_\chi(v) \neq \emptyset$ by the same argument.

Now, let $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$. We define the $\{\alpha, \beta\}$-*alternating path* P starting at u to be the maximal path consisting of alternating α and β colored edges starting at the vertex u. Note that the subgraph of G consisting of α and β colored edges consists of vertex-disjoint paths and cycles. Since α is missing at u, a path must start at u within this subgraph, so P is well defined (but may be empty, if β is also missing at v). After computing this alternating path P, the algorithm proceeds to *flip* the colors of the path P, by setting the color $\chi(e)$ of each edge e in P with color α to β, and vice versa.

The following claim shows that, after flipping the path P, we can color the edge e with β and extend the coloring χ to this edge. We note that this claim is where we crucially exploit the assumption that the graph is bipartite.

Claim 2.1.2. *After flipping the colors of P, we have that $\beta \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$.*

Proof. After flipping the path P, we have that β is now missing at u. Since β was already missing at v, it suffices to show that this is still the case after flipping the path P. This follows from the fact that v does not lie on the path P, and thus its missing colors are not affected by this operation.

To see why this is true, note that if v *does* lie on the path P, then it must be the other endpoint of P, since $\beta \in \mathsf{miss}_\chi(v)$ initially. Thus, P must be a path of even length. However, in this case, the union of P and the edge e forms an odd cycle, leading to a contradiction. □

Algorithm 1 summarizes Vizing's algorithm for bipartite graphs.

Algorithm 1: Vizing's Algorithm for Bipartite Graphs

```
Set χ(e) ← ⊥ for all e ∈ E
for each (u, v) ∈ E do
    Let α ∈ miss_χ(u) and β ∈ miss_χ(v)
    Flip the {α, β}-alternating path P starting at u
    Set χ(u, v) ← β
return χ
```

Running Time. Using standard data structures, it's easy to implement this algorithm so that the time taken to extend the coloring to an edge is proportional to the length of the alternating path flipped during the process. Since any alternating path contains each vertex at most once, it

follows that the time taken to extend the coloring to an edge is $O(n)$. Thus, the running time of this algorithm is $O(mn)$.

2.1.2 Extension to General Graphs

Unfortunately, the previous algorithm fails for general graphs since Claim 2.1.2 is no longer true. In the case that the $\{\alpha, \beta\}$-alternating path P starting at u happens to end at v, flipping the path P does not do anything useful—it only swaps which of the colors α and β are missing at u and v. In this case, we need a different way to extend the coloring χ to the edge e.

Vizing Fans and Chains. The main technical part of Vizing's algorithm is a construction that can be used to extend a $(\Delta + 1)$-coloring χ to an uncolored edge. Let $e = (u, v)$ be an uncolored edge and α be a missing color at u. This construction consists of two parts: (1) a *Vizing fan* $\mathbf{F}$, which is a sequence of colored edges incident on the vertex u, and (2) a *Vizing chain* P, which is the $\{\alpha, \gamma\}$-alternating path P starting at u, where γ is a color determined by the Vizing fan $\mathbf{F}$. Vizing showed that it is possible to extend the coloring χ to e by flipping the path P, changing the colors assigned to the edges in $\mathbf{F}$, and finally assigning a color to e (which is now missing at both endpoints).

Since the fan $\mathbf{F}$ has size $O(\Delta)$ and the chain P has length $O(n)$, this can be done in $O(n)$ time with standard data structures, leading to a running time of $O(mn)$. Crucially, we need $\Delta + 1$ colors to be able to construct the Vizing fan, so this algorithm computes a $(\Delta + 1)$-coloring. We describe the full formal construction in Section 3.2.

2.1.3 Some Simplifying Assumptions

The machinery of Vizing fans and Vizing chains that we informally described in Section 2.1.2 is a crucial part of all efficient $(\Delta + 1)$-coloring algorithms, which all invoke this construction at some point during the execution. However, the specific details of the construction of Vizing fans (see Section 3.2) are slightly technical and have a tendency to obfuscate the main conceptual ideas underpinning the algorithms, which are often relatively simple.

To improve presentation, throughout Part I, we often make the simplifying assumption that the graph is bipartite to avoid dealing with Vizing fans and chains, allowing us to convey the key ideas of the algorithms more clearly. We then explain how the algorithm can be extended to deal with general graphs—this sometimes requires some significant new insights, or can simply be a matter or inserting this machinery into the appropriate place without any additional conceptual insights.

2.2 Towards a Linear Time Vizing's Theorem

The first steps towards tackling Question 2 and obtaining a faster implementation of Vizing's theorem were independently taken by [Arj82] and [GNK+85], who showed how to obtain a running time

of $O(m\sqrt{n \log n})$. This was subsequently improved upon by [Sin19], who showed how to shave the $\sqrt{\log n}$ factor in the analysis, improving the running time to $O(m\sqrt{n})$.

These algorithms all build directly on top of Vizing's algorithm by combining it with the following two natural ideas.

Idea I: A Divide-And-Conquer Approach. The generic approach for computing a $(\Delta+1)$-coloring is to extend a partial $(\Delta+1)$-coloring of the graph one edge at a time, by recoloring some edges, until the entire graph becomes colored. It turns out that this algorithmic approach is very compatible with a simple divide-and-conquer framework for computing an edge coloring: Given a graph $G = (V, E)$ with maximum degree Δ, we can apply Eulerian partitions to divide G into two edge-disjoint subgraphs $G = G_1 \cup G_2$ with at most $\lceil m/2 \rceil$ edges and maximum degree at most $\lceil \Delta/2 \rceil$. We then find $(\lceil \Delta/2 \rceil + 1)$-colorings χ_1 and χ_2 of the subgraphs G_1 and G_2 recursively. Directly combining the colorings χ_1 and χ_2 gives a coloring χ of G with $\Delta + 3$ colors, so we have to uncolor a $2/(\Delta+3)$ fraction of the edges—amounting to $O(m/\Delta)$ edges—and extend the current partial $(\Delta+1)$-coloring χ to these edges. Thus, the 'only' remaining part is to figure out a way to color these final $O(m/\Delta)$ edges, and let the above approach take care of the rest.

Idea II: Picking Better Alternating Paths. The $O(mn)$ running time of Vizing's algorithm comes from the very pessimistic upper bound that every alternating path flipped by the algorithm might have length $O(n)$. By simply modifying the algorithm to pick which uncolored edge to color next *randomly* instead of arbitrarily, we can already get an improved bound on the running time of the algorithm.

Theorem 2.2.1. *There is an algorithm that, given any graph G on n vertices and m edges with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $O(m\Delta \log n)$ expected time.*[2]

Proof Sketch. Consider the state of the (partial) $(\Delta+1)$-coloring when λ edges remain uncolored. With some minor tweaks to the algorithm, we can ensure that each uncolored edges e corresponds to a unique alternating path (i.e. Vizing chain) P_e that is flipped if we sample and extend the coloring to e. Thus, extending the coloring to e takes $O(\Delta + |P_e|)$ time, since we spend $O(\Delta)$ time constructing the Vizing fan and $O(|P_e|)$ time constructing the Vizing chain.

We can observe that the total length of all distinct maximal alternating paths is $O(m\Delta)$, since each colored edge can only be part of Δ such paths (one for each of the Δ other colors). Thus, by sampling which uncolored edge to extend the coloring to uniformly at random, a simple averaging argument shows that the expected length of the flipped alternating path is $O(m\Delta/\lambda)$. It follows that the expected running time of the algorithm becomes $\sum_{\lambda=1}^{m} O(\Delta + m\Delta/\lambda) = O(m\Delta \log n)$. □

Following the divide-and-conquer framework, suppose we use Theorem 2.2.1 to extend a partial $(\Delta+1)$-coloring to $O(m/\Delta)$ uncolored edges. Since the algorithm can take at most $O(n)$ time to extend the coloring to some edge (since this is the maximum length of any Vizing chain), it

[2] We note that there are many different versions of this algorithm which attain the same bound using these ideas, which can even be done deterministically [GNK+85]. This proof sketch is most similar to [Sin19], and clearly demonstrates the simple averaging argument that yields the $\tilde{O}(m\Delta)$ running time.

follows that this takes $O(\min\{mn/\Delta, m\Delta \log n\}) = \tilde{O}(m\sqrt{n})$ time. Hence, the running time of the algorithm follows the recurrence $T(m, \Delta) \leq 2 \cdot T(\lceil m/2 \rceil, \lceil \Delta/2 \rceil) + \tilde{O}(m\sqrt{n})$, leading to a running time of $\tilde{O}(m\sqrt{n})$.

2.2.1 Breaking the $m\sqrt{n}$ Time Barrier

Very recently, this longstanding $\tilde{O}(m\sqrt{n})$ time barrier was broken in two independent and concurrent works by [BCC+24] and [Ass25], which improved the runtime to two incomparable bounds of $\tilde{O}(mn^{1/3})$ and $\tilde{O}(n^2)$, respectively. The techniques used in these algorithms are completely disjoint. Our algorithm in [BCC+24] follows the same framework as [GNK+85], which we highlighted above, and ultimately boils down to showing that the expected lengths of randomly sampled alternating paths are short. The main idea behind the algorithm is to ensure that the uncolored edges remain grouped in 'stars' and share some relatively small subset of vertices as endpoints, allowing us to create more Vizing chains for each uncolored edge, leading to shorter chains on average. On the other hand, the algorithm of [Ass25] takes a completely different approach based on computing matchings by performing random walks. Interestingly, these algorithms can be combined in a white-box manner to obtain a running time of $\tilde{O}(n\sqrt{m})$, by using the algorithm of [Ass25] to simplify the recursion in [BCC+24].

The Main Bottleneck. Ultimately, both of these algorithms fall short of the desired near-linear running time. In fact, we can observe that the main bottleneck in the running time for both algorithms is the same: extending the coloring to the very last few uncolored edges. This is especially obvious in the algorithm of [Ass25], which colors everything except the last $\tilde{O}(m/\Delta)$ edges in near-linear time, and then constructs $\tilde{O}(m/\Delta)$ Vizing chains to color the remaining edges, taking $\tilde{O}(n^2)$ time. Conceptually, once we are left with only $\tilde{O}(m/\Delta)$ uncolored edges, it is challenging to exploit any structure that allows us to color them efficiently, and both algorithms resort to the trivial $O(n)$ time algorithm of Vizing.

Given that the main bottleneck for these algorithms is the time it takes to individually color the last few edges one by one, a natural approach to getting further improvements is to directly tackle this bottleneck by designing a subroutine that can very efficiently extend the coloring to a single uncolored edge. To this end, suppose that we can design a subroutine that can extend a $(\Delta+1)$-coloring to an arbitrary uncolored edge in $\tilde{O}(\Delta^\gamma)$ time. Combining this subroutine with the divide-and-conquer framework, we immediately get an algorithm with a running time that follows the recurrence

$$T(m, \Delta) \leq 2 \cdot T(\lceil m/2 \rceil, \lceil \Delta/2 \rceil) + O(m/\Delta) \cdot \tilde{O}(\Delta^\gamma) = 2 \cdot T(\lceil m/2 \rceil, \lceil \Delta/2 \rceil) + \tilde{O}(m\Delta^{\gamma-1}),$$

leading to a running time of $\tilde{O}(m\Delta^{\gamma-1})$. Interestingly, we can see that a color extension subroutine that runs in $\tilde{O}(\Delta)$ would immediately yield a near-linear time algorithm. In [BCSZ25], we showed that, in combination with the techniques from preceding works [BCC+24, Ass25], such a subroutine would imply an improvement over the $\tilde{O}(mn^{1/3})$ bound of [BCC+24], for sufficiently small γ.

Lemma 2.2.2. *Let there be a $(\Delta+1)$-color extension subroutine with $\tilde{O}(\Delta^\gamma)$ runtime, for some $\gamma \geq 1$. Then there is a $(\Delta+1)$-coloring algorithm with $\tilde{O}(mn^{(\gamma-1)/(2\gamma)})$ runtime.*

Designing a Faster Color Extension Subroutine. It follows from Lemma 2.2.2 that obtaining such a color extension subroutine for $\gamma < 3$ would yield a faster $(\Delta+1)$-coloring algorithm. Incidentally, in recent years, an influential line of work [DHZ19, SV19, CHL+20, GP20, Ber22, Chr23] has addressed the very related question of finding *small augmenting subgraphs* to extend a partial coloring to an uncolored edge, primarily from a different vantage point of distributed algorithms. Many of these results were obtained using *multi-step Vizing chains*, a generalization of the central object used in Vizing's original proof [Viz64]. For a $(\Delta+1)$-color extension subroutine, the state-of-the-art running time is $\tilde{O}(\Delta^6)$, which follows directly from the dynamic algorithm of [Chr23], based on multi-step Vizing chains. It therefore remains an outstanding open question to design such a subroutine with a runtime of $\tilde{O}(\Delta^{2.99})$, as demanded by Lemma 2.2.2. By designing a new highly-optimized multi-step Vizing chain construction, building on the work of [DHZ19], we were able to implement such a subroutine with a running time of $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$, which is sufficient to get an improved running time of $\tilde{O}(mn^{1/4})$. This subroutine is a *dynamic algorithm* of independent interest, which we discuss in more detail in Section 2.3.1.

2.2.2 Hitting a Wall: The Limitations of this Approach

While this new subroutine allows us to obtain a further polynomial improvement, we are still far from the desired running time of $\tilde{O}(m)$. If we want to leverage Lemma 2.2.2 to obtain a near-linear time algorithm, then we need to design a color extension subroutine that runs in time $\tilde{O}(\Delta)$. However, as we observed earlier, such a subroutine would already immediately lead to a near-linear time algorithm, without having to exploit these recent improvements. In a sense, this indicates that these new techniques are useful for obtaining some polynomial improvement in the running time, but will not help us achieve our goal of pushing the running time all the way down to near-linear. To make matters worse, the task of designing a color extension subroutine that runs in time faster than $\tilde{O}(\Delta^2)$ seems completely out of reach. The key argument at the core of all multi-step Vizing chain constructions require the chain length to be $\Omega(\Delta^2)$, and improving beyond this would likely require developing completely different techniques for designing augmenting subgraphs.

Adopting a New Framework for Edge Coloring Algorithms: Our Approach. We now find ourselves in a difficult position—even though we seem to have made significant progress towards designing a near-linear time algorithm for $(\Delta+1)$-coloring, we also have strong reasons to believe that none of these techniques will be useful for pushing the running time all the way down to near-linear. Motivated by the inherent limitations of these techniques, in this thesis, we take a different approach to this problem. Instead of attempting to design an algorithm that extends the coloring to edges one at a time, we design an algorithm that extends the coloring to many edges simultaneously in large batches. This new approach naturally leads to a simple, combinatorial near-linear time algorithm for this problem. In Chapter 4, we overview this new framework, highlighting the key

ideas and insights of this approach. The full formal description and analysis of this near-linear time algorithm is presented in Chapter 7.

2.3 Dynamic Algorithms for Edge Coloring

In the dynamic setting, we are given a graph $G = (V, E)$ that undergoes updates via a sequence of edge insertions and deletions, while the set of vertices remains fixed. Additionally, we assume knowledge of a fixed upper bound Δ on the maximum degree of the graph of G at any point. Our task is to explicitly maintain an edge coloring χ of G as it is updated. Let $\sigma_1, ..., \sigma_\kappa$ denote the sequence of updates, and $G^{(t)} = (V, E^{(t)})$ denote the state of the graph G after the first t updates. We assume that the graph G is initially empty, i.e. $G^{(0)} = (V, \emptyset)$. Given some dynamic edge coloring algorithm, its *update time* is the time it takes to handle an update, and its *recourse* is the number of edges that change color during an update. A dynamic algorithm is capable of handling an *adaptive* (resp. *oblivious*) adversary if the update σ_t depends on (resp. does *not* depend on) the random bits used by our algorithm while handling the updates $\sigma_1, \ldots, \sigma_{t-1}$.

The Objective. Given a dynamic algorithm that maintains a μ-coloring with an update time of T, we can immediately obtain a static algorithm for computing a μ-coloring that has a running time of $m \cdot T$. This can be done by simply feeding the dynamic algorithm the entire graph one edge at a time. Naturally, the ultimate goal is to design a dynamic algorithm that 'matches' the state-of-the-art for the static setting—in the sense that one can obtain a state-of-the-art static algorithm immediately from the dynamic algorithm via this reduction. For example, in order to obtain a dynamic algorithm that matches the near-linear $O(m \log \Delta)$ time $(\Delta + 1)$-coloring algorithm in Theorem 1.1.1, we would need to obtain an update time of $O(\log \Delta)$. However, we are still far from obtaining such a result—which would settle the complexity of this problem in the dynamic setting. Consequently, the state-of-the-art is represented by various algorithms and techniques that lead to different trade offs between the number of colors needed by an algorithm and its update time.

2.3.1 Dynamic Edge Coloring via Multi-Step Vizing Chains

The first step towards designing efficient dynamic algorithms for edge coloring was made by [BM17], who showed how to maintain a $O(\Delta)$-coloring in $\tilde{O}(\sqrt{\Delta})$ update time. This was improved upon by [BCHN18], who showed how to maintain a $(2\Delta - 1)$-coloring in $O(\log \Delta)$ update time, which essentially requires dynamizing the greedy algorithm using a variant of binary search.

In order to break through the greedy threshold of $2\Delta - 1$ for dynamic edge coloring, [DHZ19] introduced the notion of *multi-step Vizing chains*, which are a concatenation of multiple (carefully chosen) Vizing chains. Suppose we have a bipartite graph G, an edge coloring χ of G, and an edge $e = (u, v)$ which is uncolored. Suppose that u and v are missing the colors α and β respectively. Then, following Vizing's algorithm Section 2.1.1, we can construct a Vizing chain P starting at u with colors α and β. In Vizing's theorem, the next step is to extend the coloring to e by flipping the colors of P, making β available at both α and β. However, if P is very long, this operation

might be prohibitively expensive. Instead, for some $1 \leq t \leq |P|$, we can *shift the uncolored edge to the t^{th} edge of the Vizing chain:* We do this by setting the color of e to β, flipping the colors of the first $t-1$ edges of P, and setting the t^{th} edge in P to be uncolored. We can then repeat this process, creating a new Vizing chain at the new uncolored edge, and shifting the uncolored edge to a new location. By carefully choosing where to create new Vizing chains and which colors to use for them, it is possible to show that this process can find a short Vizing chain and terminate very quickly (if we have sufficient colors or if Δ is sufficiently small), extending the coloring to one more uncolored edge. This process of gluing together multiple 'truncated' Vizing chains is referred to as constructing a multi-step Vizing chain.[3] The *length* of the multi-step Vizing chain is the total length of the truncated Vizing chains in this construction, i.e. the number of edges whose colors are flipped. Most of the algorithms used for constructing these multi-step Vizing chains simply make random choices about where to create new chains and which colors to use, leading to a running time which is proportional to the length of the multi-step Vizing chain.

Using this technique, [DHZ19] obtained a dynamic $(1+\epsilon)\Delta$-coloring algorithm with $O(\log^7 n/\epsilon^2)$ update time, for $\epsilon \geq \Omega(\log n/\sqrt{\Delta})$. A subsequent algorithm [Chr23], also using multi-step Vizing chains, obtained an update time of $O(\log^9 n \log^6 \Delta/\epsilon^6)$, with no restrictions on Δ. These techniques have also found many applications across other settings, such as distributed algorithms [Ber22, Chr23, BD23, Dha24] and static algorithms [BD24, BCSZ25]. Multi-step Vizing chains are particularly effective in the regime where ϵ is large or Δ is small. However, if we want to apply these techniques to the setting where ϵ is very small, such as for maintaining a $(\Delta+1)$-coloring, then these constructions are substantially less effective, leading to large poly(Δ) terms in the lengths of the multi-step Vizing chains, and also in the running times of the algorithms which exploit them. In particular, for the problem of maintaining a $(\Delta+1)$-coloring, the best update times are $\tilde{O}(\Delta^6)$ by [Chr23], or the trivial $O(n)$ of [Viz64].

A Lower Bound for Extending Edge Colorings. A multi-step Vizing chain for an uncolored edge e is a specific instance of a more general object, known as an *augmenting subgraph*, which is a subset of edges H such that we can extend the coloring to the edge e by only changing the colors of edges in H. A lower bound by [CHL+20] shows that there exists a graph G and a $(1+\epsilon)\Delta$-coloring χ with an uncolored edge e such that extending the coloring χ to e requires an augmenting subgraph of size $\Omega(\log(\epsilon n)/\epsilon)$, when $\epsilon \leq 1/3$. Consequently, for $(\Delta+1)$-coloring, any multi-step Vizing chain construction must have length $\tilde{\Omega}(\Delta)$.

Faster Dynamic $(\Delta+1)$-Coloring via Multi-Step Vizing Chains. Given the broad applicability of these constructions, and the large gap between the upper and lower bounds for $(\Delta+1)$-coloring, obtaining improved multi-step Vizing chains for $(\Delta+1)$-coloring is an interesting research direction. By building on the work of [DHZ19], we were able to design a new multi-step Vizing chain construction, leading to a dynamic $(\Delta+1)$-coloring algorithm with an update time of $\tilde{O}(\Delta^2+\sqrt{\Delta n})$, which improves upon the state-of-the-art for a large range of Δ. As discussed in Section 2.2.1, we used this dynamic algorithm in combination with previous works [BCC+24, Ass25] to obtain faster

[3] This term was coined by [Ber22], although similar ideas were considered independently by [DHZ19].

static algorithms for $(\Delta+1)$-coloring [BCSZ25]. We present this algorithm in Chapters 5 and 9 as one of the contributions of this thesis.

2.3.2 The Quest for Constant Update Time

Achieving constant update time for fundamental problems is an important research agenda within dynamic algorithms [AS21, BCH17, BGK+22, BGM17, BHNW21, BK19, HP20, PS16, Sol16, SW18]. There are two major considerations that underpin this research agenda. (i) A constant update time algorithm rules out the possibility of obtaining a (cell-probe) lower bound for the concerned problem [Lar12, PD06]. (ii) It immediately implies a *linear time* algorithm for the concerned problem in the static setting, and thus aligns with what is essentially the best possible static guarantee. Thus, for the regime where ϵ is a small fixed constant, can we obtain constant update time? With this backdrop, we encounter a significant hurdle at the very beginning of this quest, since currently there does not even exist a $O_\epsilon(m)$ time *static* algorithm for $(1+\epsilon)\Delta$-coloring.[4] In fact, if we insist upon getting an exact linear (i.e., $O(m)$) running time, and subject to this constraint try to minimize the number of colors being used, then the only game in town happens to be a very simple, folklore randomized greedy algorithm that gives us $(2+\epsilon)\Delta$-coloring. This algorithm can also be dynamized to get $O(1/\epsilon)$ update time.

Additionally, an implication of the lower bound discussed in Section 2.3.1 is that any dynamic algorithm which is analyzed in a 'memoryless' manner, assuming the edge coloring is arbitrary at the start of each update, *must* have a recourse—and thus also an update time—of $\Omega(\log(\epsilon n)/\epsilon)$. All dynamic algorithms based on Vizing chains are analyzed in this memoryless manner [DHZ19, Chr23, Chr26], and incur a large polylogarithmic factor in their update times. In order to design a dynamic algorithm with an update time of $\text{poly}(1/\epsilon)$, we must take a different approach that allows us to bypasses this lower bound.

Optimal Dynamic $(1+\epsilon)\Delta$-Coloring via the Nibble Method. If we only care about the recourse of the solution (and ignore the update time), then there exists a $(1+\epsilon)\Delta$-coloring algorithm with $\text{poly}(1/\epsilon)$ recourse based on the Nibble method [BGW21], that was first used in the context of edge coloring in the distributed setting [DGP98]. It seems very difficult to implement the algorithm of [BGW21] using $O_\epsilon(1)$ update time data structures, for two reasons. First, their dynamic algorithm needs to perform expensive resampling after every update, and it is not at all clear how to implement this resampling efficiently, i.e., in constant update time. Second, and more fundamentally, all existing Nibble method-based algorithms require some form of *regularization gadget*, since the inductive approach that is used to analyze these algorithms does not work on graphs that are not near-regular. Implementing this gadget requires $\Omega(n\Delta^2)$ running time in the static setting, and $\Omega(n\Delta^2)$ preprocessing time in the dynamic setting.

Building on this approach, we design a new variant of the Nibble method, which does not require the input graph to be regular and can also be dynamized efficiently, leading to a dynamic $(1+\epsilon)\Delta$-coloring algorithm with $\text{poly}(1/\epsilon)$ update time, for $\Delta \geq \tilde{\Omega}(1)$. This also leads to the first non-trivial

[4] We use $O_\epsilon(\cdot)$ to hide $\text{poly}(1/\epsilon)$ factors.

linear time static algorithm for this problem, achieving a running time of $O(m \operatorname{poly}(1/\epsilon))$. We present this algorithm in Chapters 5 and 10 as one of the contributions of this thesis.

Follow-Up Works on Linear Time Edge Coloring. A follow-up work by [BD24] obtained a static algorithm with $O(m \operatorname{poly}(1/\epsilon))$ running time with no restriction on Δ. More recently, [Ass25] gave a static algorithm with exponentially better dependence on $1/\epsilon$, obtaining a running time of $O(m \log(1/\epsilon))$ when $\Delta \geq \omega(\log n/\epsilon)$. These algorithms use completely different techniques and do not extend to the dynamic setting.

Chapter 3

Technical Background

In this chapter, we provide the basic technical background and preliminaries for this thesis. In Section 3.1, we describe the basic notation used throughout the thesis. In Section 3.2, we formally describe the notion of Vizing chains and Vizing fans.

3.1 Basic Notation

Let $G = (V, E)$ be graph on n vertices with m edges and maximum degree Δ and let $\chi : E \longrightarrow C \cup \{\perp\}$ be a (partial) $|C|$-coloring of G. We refer to edges $e \in E$ with $\chi(e) = \perp$ as *uncolored*. We sometimes abbreviate 'partial coloring' by 'coloring' when it is clear that edges can be uncolored. Given a vertex $u \in V$, we denote the set of colors that are not assigned to any edge incident on u by $\mathsf{miss}_\chi(u)$. We sometimes refer to $\mathsf{miss}_\chi(u)$ as the *palette* of u. We say that the colors in $\mathsf{miss}_\chi(u)$ are *missing* (or *available*) at u.

Given a path $P = e_1, \ldots, e_k$ in G, we say that P is an $\{\alpha, \beta\}$-*alternating path* if $\chi(e_i) = \alpha$ whenever i is odd and $\chi(e_i) = \beta$ whenever i is even (or vice versa). We say that the alternating path P is *maximal* if one of the colors α or β is missing at each of the endpoints of P. We refer to the process of changing the color of each edge $e_i \in P$ with color α (resp. β) to β (resp. α) as *flipping* the path P. We denote by $|P|$ the length (i.e., the number of edges) of the alternating path P. We define the *length i prefix* of the path P to be the path $P_{\leq i} := e_1, \ldots, e_i$.

Consider a set $U \subseteq E$ of edges that are uncolored under χ, i.e., $\chi(e) = \perp$ for all $e \in U$. We use the phrase *'extending χ to U'* to mean the following: Modify χ so as to ensure that $\chi(e) \neq \perp$ for all $e \in U$, without creating any new uncolored edges. When the set U consists of a single edge e (i.e., when $U = \{e\}$), we use the phrase *'extending χ to the edge e'* instead of *'extending χ to U'*.

Our algorithms will always work by modifying a partial coloring χ; unless explicitly specified otherwise, every new concept we define (such as u-fans and separable collection in Section 7.1) will be defined with respect to this particular partial coloring χ.

Remark. Our algorithms based on the Nibble method, which we overview in Sections 5.3 to 5.5 and present formally in Chapter 10, use different techniques to rest of our algorithms and thus also

use different notation, which is defined in the relevant sections and chapters.

3.2 Vizing Fans and Vizing Chains

We now define the notion of *Vizing fans*, that we informally described in Section 2.1.2, which has been used extensively in the edge coloring literature [Viz64, GNK+85, Chr23, Sin19, BCC+24].

Definition 3.2.1 (Vizing fan)**.** *A* Vizing fan *is a sequence* $\mathbf{F} = (u, \alpha), (v_1, c_1), \ldots, (v_k, c_k)$ *where* $u, v_1, \ldots, v_k$ *are distinct vertices and* $c_1, \ldots, c_k$ *are colors such that*

1. $\alpha \in \mathsf{miss}_\chi(u)$ *and* $c_i \in \mathsf{miss}_\chi(v_i)$ *for all* $i \in [k]$.
2. $v_1, \ldots, v_k$ *are distinct neighbours of* u.
3. $\chi(u, v_1) = \perp$ *and* $\chi(u, v_i) = c_{i-1}$ *for all* $i > 1$.
4. *Either* $c_k \in \mathsf{miss}_\chi(u)$ *or* $c_k \in \{c_1, \ldots, c_{k-1}\}$.

We say that the Vizing fan $\mathbf{F} = (u, \alpha), (v_1, c_1), \ldots, (v_k, c_k)$ is α-*primed*, has *center* u and *leaves* $v_1, \ldots, v_k$. We refer to c_i as the color of v_i within $\mathbf{F}$. A crucial property is that we can *rotate* colors around the Vizing fan $\mathbf{F}$ by setting $\chi(u, v_1) \leftarrow c_1, \ldots, \chi(u, v_{i-1}) \leftarrow c_{i-1}, \chi(u, v_i) \leftarrow \perp$ for any $i \in [k]$. We say that $\mathbf{F}$ is a *trivial* Vizing fan if $c_k \in \mathsf{miss}_\chi(u)$. Note that, if $\mathbf{F}$ is trivial, we can immediately extend the coloring χ to (u, v_1) by rotating all the colors around $\mathbf{F}$ and setting $\chi(u, v_k) \leftarrow c_k$.

Algorithm 2 describes the standard procedure used to construct Vizing fans. As input, it takes a vertex u and a color $\alpha \in \mathsf{miss}_\chi(u)$, and returns an α-primed Vizing fan with center u.

Algorithm 2: `Vizing-Fan`(u, v, α)

```
For each x ∈ V, let clr(x) ∈ miss_χ(x)
k ← 1 and v_1 ← v
c_1 ← clr(v_1)
while c_k ∉ {c_1, ..., c_{k-1}} and c_k ∉ miss_χ(u) do
    Let (u, v_{k+1}) be the edge with color χ(u, v_{k+1}) = c_k
    c_{k+1} ← clr(v_{k+1})
    k ← k + 1
return (u, α), (v_1, c_1), ..., (v_k, c_k)
```

The Algorithm Vizing**.** We now describe the algorithm Vizing that, given a Vizing fan $\mathbf{F} = (u, \alpha), (v_1, c_1), \ldots, (v_k, c_k)$ as input, extends the coloring χ to the edge (u, v_1) by building a *Vizing chain*. Algorithm 3 gives a formal description of this procedure.

Algorithm 3: Vizing($\mathbf{F}$)

```
if F is trivial then
    χ(u, v_i) ← c_i for all i ∈ [k]
    return
Let P denote the maximal {α, c_k}-alternating path starting at u
Extend χ to (u, v_1) by flipping the path P and rotating colors in F (details in Lemma 3.2.2)
```

Thus, running Vizing($\mathbf{F}$) extends the coloring χ to the uncolored edge (u, v_1) by rotating colors in the Vizing fan $\mathbf{F}$ and flipping the colors of the alternating path P. The alternating path P is often referred to as a *Vizing chain*. We sometimes refer to the process of running Vizing($\mathbf{F}$) as *activating* the Vizing fan $\mathbf{F}$.

Lemma 3.2.2. *Algorithm 3 extends the coloring χ to the edge (u, v_1) in time $O(\Delta + |P|)$.*

Proof. To see that the path P is well-defined, note that $\alpha \in \mathsf{miss}_\chi(u)$ and $c_k \notin \mathsf{miss}_\chi(u)$, so there is an $\{\alpha, c_k\}$-alternating path starting at u.

Extending the coloring χ: By the definition of a Vizing fan, c_k is the first repetition of some color in $\{c_1, \ldots, c_{k-1}\}$, so there is a unique index $j \in [k-1]$ such that $c_j = c_k$, and P has (u, v_{j+1}) as its first edge with color c_j. We consider the cases where the path P does or does not have v_j as an endpoint. If P does not end at v_j, then we can rotate the colors of the first $j+1$ edges in the fan by setting $\chi(u, v_1) \leftarrow c_1, \ldots, \chi(u, v_j) \leftarrow c_j$ and flip the colors of the alternating path P. If P does end at v_j, then we flip the colors of the alternating path P, rotate the colors of the fan by setting $\chi(u, v_1) \leftarrow \chi(u, v_2), \ldots, \chi(u, v_{k-1}) \leftarrow \chi(u, v_k)$, and set $\chi(u, v_k) \leftarrow c_k$. Note that, while rotating the fan in this last step, we have $\chi(u, v_{j+1}) = \alpha \neq c_k$.

Using standard data structures (see Section 7.5), we can implement this algorithm to run in time proportional to the number of edges that change their colors, which is $O(\Delta + |P|)$. □

Given a Vizing fan $\mathbf{F}$, we denote the path P (the Vizing chain) considered by Algorithm 3 by `Vizing-Path`($\mathbf{F}$). If the Vizing fan $\mathbf{F}$ is trivial, then `Vizing-Path`($\mathbf{F}$) denotes an empty path $\emptyset$.

Remark. Our algorithm for constructing multi-step Vizing chains, which we overview in Sections 5.1 and 5.2 and present formally in Chapter 9, uses different notation for Vizing fans and chains for notational convenience, which we describe in Section 9.1.

Chapter 4

Our Algorithms for Static Edge Coloring

In this chapter, we provide a detailed technical overview of our static algorithms for $(\Delta+1)$-coloring, highlighting our key ideas and techniques. In Section 4.1, we overview our randomized near-linear time algorithm (Theorem 1.1.1). In Section 4.2, we instantiate our near-linear time algorithm on bipartite graphs to showcase some of our key insights. In Section 4.3, we overview our deterministic almost-linear time algorithm (Theorem 1.1.2).

Notation. This chapter uses the notation described in Chapter 3. Any other definitions or notation are introduced throughout the chapter.

4.1 Vizing's Theorem in Randomized Near-Linear Time

In this section, we provide an overview of our randomized near-linear time algorithm for $(\Delta+1)$-coloring, which resolves the time complexity of this problem up to a $O(\log \Delta)$ factor. We summarize this algorithm in Theorem 1.1.1, which we restate below. We give the full details of this algorithm in Chapter 7.

Theorem 1.1.1. *There is an algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $O(m \log \Delta)$ time with high probability.*

Our main contribution in this work is to improve the time-complexity of $(\Delta+1)$-edge coloring by polynomial factors all the way to near-linear. For this reason, as well as for the sake of transparency of our techniques, we focus primarily on presenting a $\tilde{O}(m)$ time randomized algorithm, which showcases our most novel ideas. Later in Chapter 7, we show how to optimize the polylogarithmic factors to obtain a clean $O(m \log \Delta)$ runtime.

4.1.1 Overview of Our Randomized Algorithm

For the sake of completeness, we now present a self-contained overview of prior approaches to $(\Delta+1)$-coloring and describe our techniques at a high level; see Chapter 2 for a more detailed overview of prior approaches and their limitations. For the following discussions, we will assume

basic familiarity with Vizing's proof and its underlying algorithm; see Section 3.2 for more details on Vizing's algorithm.

Prior Approaches. A generic approach for $(\Delta+1)$-edge coloring, dating back to Vizing's proof itself, is to extend a partial $(\Delta+1)$-edge coloring of the graph one edge at a time, possibly by recoloring some edges, until the entire graph becomes colored. As expected, the main bottleneck in the runtime of this approach comes from extending the coloring to the last few uncolored edges. For instance, given a graph $G=(V,E)$ with maximum degree Δ, we can apply Eulerian partitions to divide G into two edge-disjoint subgraphs $G=G_1\cup G_2$ with maximum degrees at most $\lceil\Delta/2\rceil$. We then find $(\lceil\Delta/2\rceil+1)$-edge colorings of the subgraphs G_1 and G_2 recursively. Directly combining the colorings of G_1 and G_2 gives a coloring of G with $\Delta+3$ colors, so we have to uncolor a $2/(\Delta+3)$ fraction of the edges—amounting to $O(m/\Delta)$ edges—and try to extend the current partial $(\Delta+1)$-edge coloring to these edges. Thus, the "only" remaining part is to figure out a way to color these final $O(m/\Delta)$ edges, and let the above approach take care of the rest.

This task of extending the coloring to the last $\Theta(m/\Delta)$ edges is the common runtime bottleneck of all previous algorithms. Vizing's original algorithm [Viz64] gives a procedure to extend any partial coloring to an arbitrary uncolored edge (u,v) by rotating some colors around u and flipping the colors of an alternating path starting at u. The runtime of this procedure would be proportional to the size of the rotation, usually called a *Vizing fan*, and the length of the alternating path, usually called a *Vizing chain*, which are bounded by Δ and n respectively. As such, the total runtime for coloring the remaining $O(m/\Delta)$ edges using Vizing fans and Vizing chains will be $O(mn/\Delta)$ time.

As one can see, flipping long alternating paths is the major challenge in Vizing's approach. To improve this part of the runtime, [GNK+85] designed an algorithm that groups all uncolored edges into $O(\Delta^2)$ types depending on the two colors of the Vizing chain induced by this edge. Since all the Vizing chains of the same type are vertex-disjoint, they can be flipped simultaneously and their total length is only $O(n)$. This means that the runtime of coloring all edges of a single type can be bounded by $O(n)$ as well. This leads to an $O(n\Delta^2)$ time algorithm for handling all $O(\Delta^2)$ types; a more careful analysis can bound this even with $O(m\Delta)$ time. Finally, balancing the two different bounds of $O(mn/\Delta)$ and $O(m\Delta)$ yields a runtime bound of $O(m\sqrt{n})$ for coloring $O(m/\Delta)$ edges, which leads to an $\tilde{O}(m\sqrt{n})$ time algorithm using the above framework.

There has been some very recent progress that broke through this classical barrier of $O(m\sqrt{n})$ time in [GNK+85, Arj82]. In [BCC+24], the authors speed up the extension of the coloring to uncolored edges when these edges admit a small vertex cover. They then show how to precondition the problem so that uncolored edges admit a small vertex cover, leading to a $\tilde{O}(mn^{1/3})$ time algorithm. In [Ass25], the author avoided the need for Eulerian partition and recursion altogether by instead designing a new near-linear time algorithm for $(\Delta+O(\log n))$-edge coloring. This algorithm borrows insights from sublinear matching algorithms in regular bipartite graphs by [GKK10] and is thus completely different from other edge coloring algorithms mentioned above. By using this algorithm, finding a $(\Delta+1)$-edge coloring directly reduces to the color extension problem with $O((m\log n)/\Delta)$ uncolored edges (by removing the colors of a $\Theta(\log n)/\Delta$ fraction of the edges in

the $(\Delta + O(\log n))$-edge coloring to obtain a partial $(\Delta+1)$-edge coloring first). Applying Vizing's procedure for these uncolored edges takes additional $\tilde{O}(mn/\Delta) = \tilde{O}(n^2)$ time, leading to an $\tilde{O}(n^2)$ time algorithm for $(\Delta+1)$-edge coloring. Finally, in [BCSZ25], the authors showed that a $(\Delta+1)$-coloring can be computed in $\tilde{O}(mn^{1/4})$ time, by using the algorithm of [Ass25] for initial coloring of the graph and then presenting an improved color extension subroutine with a runtime of $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$ for coloring each remaining edge; the best previous color extension time bounds were either the trivial $O(n)$ bound or the bound $\tilde{O}(\Delta^4)$ by [Ber22, BD24].

Our Approach: A Near-Linear Time Color Extension Algorithm

We will no longer attempt to design a faster color extension for a *single edge*, and instead color them in large *batches* like in [GNK+85], which allows for a much better amortization of runtime in coloring multiple edges. This ultimately leads to our main technical contribution: a new randomized algorithm for solving the aforementioned color extension problem for the last $O(m/\Delta)$ edges in $\tilde{O}(m)$ time. With this algorithm at hand, we can follow the aforementioned Eulerian partition approach and obtain a $(\Delta+1)$-edge coloring algorithm whose runtime $T(m)$ follows the recursion $T(m) \leq 2T(m/2) + \tilde{O}(m)$ with high probability; this implies that $T(m) = \tilde{O}(m)$, hence, giving us a near-linear time randomized algorithm for $(\Delta+1)$-edge coloring. This way, we will not even need to rely on the $(\Delta + O(\log n))$-edge coloring algorithm of [Ass25] to color the earlier parts of the graph (although one can use that algorithm instead of Eulerian partition approach to the same effect).

We now discuss the main ideas behind our color extension algorithm. In the following, it helps to think of the input graph as being near-regular (meaning that the degree of each vertex is $\Theta(\Delta)$), and thus the total number of edges will be $m = \Theta(n\Delta)$; this assumption is *not* needed for our algorithm and is only made here to simplify the exposition.

Color Type Reduction. Recall that the runtime of $O(n\Delta^2)$ for the color extension algorithm of [GNK+85] is due to the fact that there are generally $O(\Delta^2)$ types of alternating paths in the graph and that the total length of the paths of each color type is bounded by $O(n)$ edges. However, *if* it so happens that the existing partial coloring only involves $O(\Delta)$ color types instead, then the same algorithm will only take $O(n\Delta) = O(m)$ time (by the near-regularity assumption). The underlying idea behind our algorithm is to modify the current partial edge coloring (without decreasing the number of uncolored edges) so that the number of color types reduces from $O(\Delta^2)$ to $O(\Delta)$ only.

To explore this direction, let us **assume for now the input graph is bipartite**, which greatly simplifies the structure of Vizing fans and Vizing chains, thus allowing us to convey the key ideas more clearly (see Section 4.2 for a more detailed exposition); later we highlight some of the key challenges that arise when dealing with general graphs. We shall note that it has been known since the 80s that one can Δ-edge color bipartite graphs in $\tilde{O}(m)$ time [CH82, COS01]. However, the algorithms for bipartite graphs use techniques that are entirely different from Vizing fans and Vizing chains, and which do not involve solving the color extension problem at all. In particular, prior to this work, it was unclear whether one can efficiently solve the color extension problem in bipartite graphs. Therefore, the assumption of a bipartite input graph does not trivialize our goal of using Vizing fans and Vizing chains for efficiently solving the color extension problem. Additionally,

we can assume that in the color extension problem, the last $O(m/\Delta)$ edges to be colored can be partitioned into $O(1)$ matchings (this guarantee follows immediately from the recursive framework we outlined earlier), and that we deal with each of these matchings separately. In other words, we can also **assume that the uncolored edges are vertex-disjoint**.

Let χ be a partial $(\Delta+1)$-edge coloring, and for any vertex $w \in V$, let $\mathsf{miss}_\chi(w) \subseteq [\Delta+1]$ be the set of colors missing from the edges incident to w under χ. Given any uncolored edge (u,v), the color type of this edge (u,v) would be $\{c_u, c_v\}$ for some arbitrary choices of $c_u \in \mathsf{miss}_\chi(u)$ and $c_v \in \mathsf{miss}_\chi(v)$ (it is possible for an edge to be able to choose more than one color type, but we fix one arbitrary choice among them); in other words, if we flip the $\{c_u, c_v\}$-alternating path starting at u, then we can assign $\chi(u,v)$ to be c_v. To reduce the total number of different color types to $O(\Delta)$, we would have to make some color types much more *popular*: at the beginning, a type spans an $\Omega(1/\Delta^2)$ proportion of the uncolored edges but we would like to have a type spanning an $\Omega(1/\Delta)$ proportion. For this purpose, we fix an arbitrary color type $\{\alpha, \beta\}$, and want to modify χ to transform the type of an arbitrary uncolored edge (u,v) from $\{c_u, c_v\}$ to $\{\alpha, \beta\}$ – we call this *popularizing* the edge (u,v). To do this, we can simply flip the $\{\alpha, c_u\}$-alternating path P_u starting at u and the $\{\beta, c_v\}$-alternating path P_v starting at v.

There are two technical issues regarding this path-flipping approach. Firstly, the alternating paths P_u and P_v could be very long, and require a long time for being flipped. More importantly, flipping P_u and P_v could possibly damage other $\{\alpha, \beta\}$-type (uncolored) edges that we popularized before. More specifically, say that we have popularized a set Φ of uncolored edges. When popularizing the next uncolored edge (u,v), it could be the case that the $\{\alpha, c_u\}$-alternating path P_u is ending at a vertex u' for some edge $(u', v') \in \Phi$. If we flip the path P_u, then (u', v') would no longer be of $\{\alpha, \beta\}$-type as α would not be missing at u' anymore. See Figure 4.1 for an illustration.

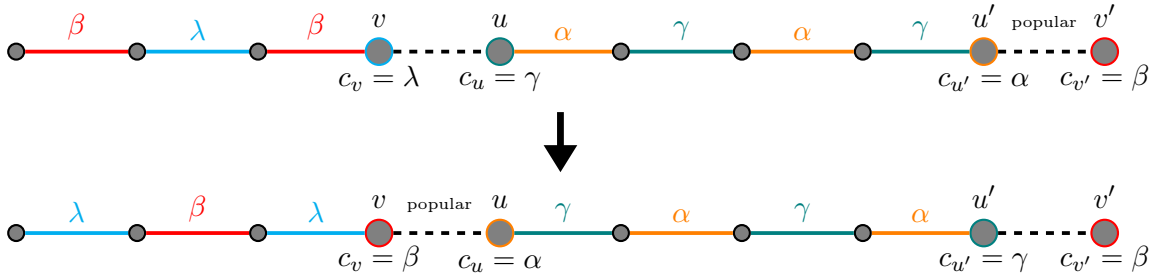

Figure 4.1: In this picture, we attempt to popularize edge (u,v) by flipping the $\{\alpha, \gamma\}$-alternating path from u and the $\{\beta, \lambda\}$-alternating path from v. However, flipping the $\{\alpha, \gamma\}$-alternating path from u makes a previously popular edge (u', v') unpopular as u' will not miss color α anymore.

Our key observation is that when $|\Phi|$ is relatively small, most choices for an alternating path P_u cannot be ending at edges in Φ. Consider the above bad example where P_u is a $\{c_u, \alpha\}$-alternating path ending at u' for some $(u', v') \in \Phi$. Let us instead look at this from the perspective of the $\{\alpha, \beta\}$-type edge $(u', v') \in \Phi$. For any $(u', v') \in \Phi$ and any color γ, there can be at most one uncolored edge $(u,v) \notin \Phi$ whose corresponding path P_u is the same $\{\gamma, \alpha\}$-alternating path starting

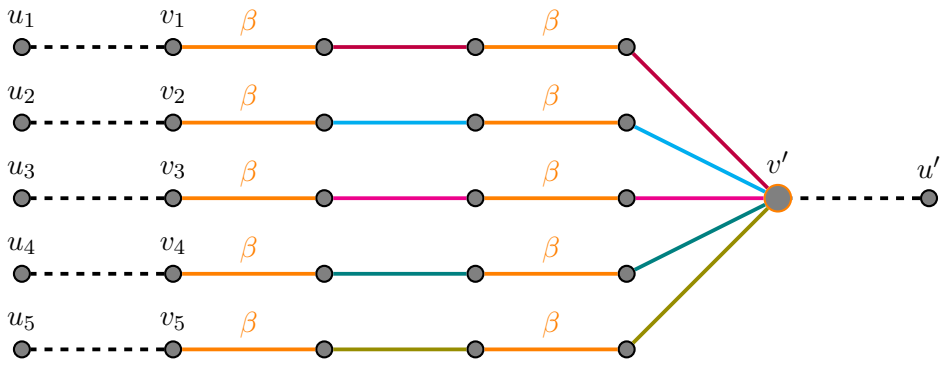

Figure 4.2: In this picture, $(u', v') \in \Phi$ with $c_{u'} = \alpha, c_{v'} = \beta$. For each uncolored edge (u_i, v_i), flipping the $\{c_i, \beta\}$-alternating path from v_i would damage the property that $c_{v'} = \beta$. Fortunately, there are at most Δ many different such (u_i, v_i) because each of them is at the end of an $\{\beta, \cdot\}$-alternating path starting at v'.

at u'; this is because any vertex belongs to at most one alternating path of a fixed type (here, the type $\{\gamma, \alpha\}$) and each vertex belongs to at most one uncolored edge (recall that the uncolored edges form a matching). This is also true for $\{\gamma, \beta\}$-type edges for the same exact reason. As such, ranging over all possible choices of $\gamma \in [\Delta + 1]$, there are at most $O(|\Phi|\Delta)$ uncolored edges (u, v) whose alternating paths could damage the set Φ. Therefore, as long as the size of Φ is a $o(1/\Delta)$ fraction (or more precisely, an $O(1/\Delta)$ fraction, for a sufficiently small constant hiding in the O-notation) of all uncolored edges, the following property holds: a constant fraction of uncolored edges $e \notin \Phi$ can be popularized using the above method without damaging Φ. See Figure 4.2 for an illustration.

This property resolves both technical issues raised earlier simultaneously. Let λ denote the number of uncolored edges. For the second issue, as long as $|\Phi| = o(\lambda/\Delta)$, we can take a random uncolored edge $(u, v) \notin \Phi$ and flip P_u and P_v if this does not damage any already popularized edge in Φ; by the observation above, this happens with constant probability. For the first issue, we can show that the expected length of alternating paths P_u and P_v, for the random uncolored edge picked above, is $O(m/\lambda)$; indeed, this is because the number of edges colored α or β is $O(n)$, hence the total length of all alternating paths with one color being fixed to α or β is $O(m)$. All in all, the total time we spend to popularize a single type $\{\alpha, \beta\}$ to become a $\Theta(1/\Delta)$ fraction of all uncolored edges is $O(\lambda/\Delta \cdot m/\lambda) = O(m/\Delta)$. Since coloring edges of a single type takes $O(n) = O(m/\Delta)$ time by our earlier discussion, we can color in this way a $\Theta(1/\Delta)$ fraction of all uncolored edges in $O(m/\Delta)$ expected time. As a direct corollary, we can color all uncolored edges in $O(m \log n)$ time, hence solving the color extension problem, in (near-regular) bipartite graphs, in near-linear time. The above argument will be detailed in the proof of Lemma 4.2.2 in Section 4.2 (see also Lemma 7.2.3 of Section 7.4 for the analogous but not identical argument in general graphs).

Collecting U-Fans in General Graphs. We now discuss the generalization of our scheme above to non-bipartite graphs. The existence of odd cycles in non-bipartite graphs implies that we can no longer assign a color type $\{c_u, c_v\}$ to an uncolored edge (u, v) for $c_u \in \mathsf{miss}_\chi(u), c_v \in \mathsf{miss}_\chi(v)$, and hope that flipping the $\{c_u, c_v\}$-alternating path from u allows us to color the edge (u, v) with c_v

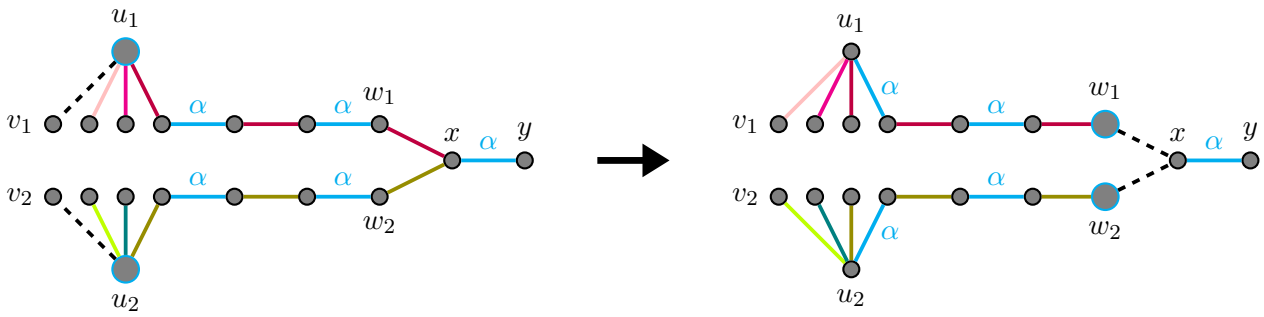

Figure 4.3: In this picture, we have two different uncolored edges $(u_1, v_1), (u_2, v_2)$ such that $\alpha \in \mathsf{miss}_\chi(u_1) \cap \mathsf{miss}_\chi(u_2)$, and their Vizing chains first intersect at edge (x, y) which currently has color α under χ. Then we can rotate their Vizing fans and flip part of their Vizing chains to shift $(u_1, v_1), (u_2, v_2)$ to $(w_1, x), (w_2, x)$ respectively to form a u-fan; note that $\alpha \in \mathsf{miss}_\chi(w_1) \cap \mathsf{miss}_\chi(w_2)$ after this shifting procedure.

(because the path may end at v, and thus after flipping it c_v will no longer be missing from v). This is where Vizing fans and Vizing chains come into play: in non-bipartite graphs, a color type of an uncolored edge (u, v) is $\{c_u, \gamma_{u,v}\}$ where c_u is an arbitrary color in $\mathsf{miss}_\chi(u)$ but $\gamma_{u,v}$ is determined by the Vizing fan around u and the alternating path that we take for coloring (u, v) (or the Vizing chain of (u, v)). Thus, while as before we can switch c_u with some fixed color α, it is unclear how to flip alternating paths to change $\gamma_{u,v}$ to β also, in order to popularize the edge (u, v) to be of some designated type $\{\alpha, \beta\}$, without damaging another popularized edge as a result.

To address this challenge, we rely on the notion of a *u-fan*, introduced by [GNK+85], which is the non-bipartite graph analogue of an uncolored edge in bipartite graphs. A u-fan of type $\{\alpha, \beta\}$ is a pair of uncolored edges $(u, v), (u, w)$ such that $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$. Consider the $\{\alpha, \beta\}$-alternating path starting at u. As at least one of v or w is not the other endpoint of this alternating path (say v), flipping this path allows us to assign the color β to (u, v). Consequently, as u-fans are similar to edges in bipartite graphs, we can still essentially (but not exactly) apply our color type reduction approach if all the uncolored edges are paired as u-fans. Therefore, it suffices to modify χ to pair all uncolored edges together to form u-fans.

In order to pair different uncolored edges together and form u-fans, we should first be able to move uncolored edges around. Such operations already appeared in some previous work on dynamic edge coloring [DHZ19, Chr23, Chr26]. Basically, for any uncolored edge (u, v), we can modify χ to shift this uncolored edge to any position on its Vizing chain. This naturally leads to a win-win situation: if the Vizing chain is short, then (u, v) can be colored efficiently using Vizing's procedure; otherwise if most Vizing chains are long, then there must be a pair of Vizing chains meeting together after a few steps, so we can shift two uncolored edges to form a u-fan efficiently.

Let us make this a bit more precise. Fix an arbitrary color α and consider the set U_α of all the uncolored edges (u, v) such that $\alpha \in \mathsf{miss}_\chi(u)$ and the respective Vizing chain is of type $\{\alpha, \cdot\}$. Also, assume there are m_α edges colored α under χ. If most $\{\alpha, \cdot\}$-Vizing chains have length larger than

$\Omega(m_\alpha/|U_\alpha|)$, then on average, two Vizing chains should meet within the first $O(m_\alpha/|U_\alpha|)$ steps; in this case, we can repeatedly pick two intersecting Vizing chains and create a u-fan by shifting their initial uncolored edges to the intersection of these Vizing chains; see Figure 4.3 for illustration. Given the length of the chains, this takes $O(m_\alpha/|U_\alpha|)$ time. Otherwise, the average cost of applying Vizing's color extension procedure is $O(m_\alpha/|U_\alpha|)$, and in this case we can directly color all those edges in $O(m_\alpha)$ time. Summing over all different $\alpha \in [\Delta+1]$ gives a near-linear runtime. The above argument will be detailed in the proof of Lemma 7.2.2 in Section 7.3.

The above discussion leaves out various technical challenges. For instance, moving around uncolored edges as described above breaks the assumption that the uncolored edges form a matching. Handling this requires dedicating *different* colors from $\mathsf{miss}_\chi(u)$ for every uncolored edge incident on a vertex u. This is formalized via the notion of *separability* in Section 7.1.1. Additionally, we have ignored all algorithmic aspects of (efficiently) finding pairs of intersecting Vizing chains, as well as the corner cases of Vizing fan intersections and fan-chain intersections. We defer the discussions on these details to the actual proofs in subsequent sections.

4.2 Showcase: Our Randomized Algorithm on Bipartite Graphs

In this section, we instantiate our algorithm on bipartite graphs to showcase some of our key insights, and outline a proof of Theorem 4.2.1 below.

Theorem 4.2.1. *There is a randomized algorithm that, given a bipartite graph $G = (V, E)$ with maximum degree Δ, returns a Δ-edge coloring of G in $\tilde{O}(m)$ time with high probability.*

As explained in Section 4.1.1, focusing on bipartite graphs allows us to ignore the technical issues that arise while dealing with Vizing fans. At the same time, this does *not* trivialize the main conceptual ideas underpinning our algorithm. In particular, we prove Theorem 4.2.1 via Lemma 4.2.2 below, which gives a specific *color-extension* algorithm on bipartite graphs. Although near-linear time Δ-edge coloring algorithms on bipartite graphs existed since the 1980s [CH82, COS01], to the best of our knowledge there was no known algorithm for Lemma 4.2.2 prior to our work.

Lemma 4.2.2. *Let $\chi : E \to [\Delta] \cup \{\perp\}$ be a partial Δ-edge coloring in a bipartite graph $G = (V, E)$, and let $U \subseteq E$ be a matching of size λ such that every edge $e \in U$ is uncolored in χ. Furthermore, suppose that we have access to an "auxiliary data structure", which allows us to detect in $\tilde{O}(1)$ time the two least common colors $\alpha, \beta \in [\Delta]$ in χ.[1] Then there is a randomized algorithm that can extend χ to $\Omega(\lambda/\Delta)$ many edges of U in $\tilde{O}(m/\Delta)$ time with high probability.*

Let us start by showing that Lemma 4.2.2 implies Theorem 4.2.1 easily.

Proof of Theorem 4.2.1. We follow the strategy outlined in Section 4.1.1. Given a bipartite graph G, we first find an Eulerian partition of the graph to partition the edges of G into two subgraphs

[1] Specifically, for every $\gamma \in [\Delta] \setminus \{\alpha, \beta\}$ and $\gamma' \in \{\alpha, \beta\}$, we have $|\{e \in E \mid \chi(e) = \gamma\}| \geq |\{e \in E \mid \chi(e) = \gamma'\}|$.

of maximum degree $\lceil \Delta/2 \rceil$ each, and color them recursively using different colors. This leads to a $2 \cdot \lceil \Delta/2 \rceil \leq (\Delta + 2)$ edge coloring of G. We then form a partial Δ edge coloring χ by uncoloring the two color classes of this $(\Delta + 2)$-edge coloring with the fewest edges assigned to them, which leaves us with two edge-disjoint matchings U_1 and U_2 to color. This is our previously mentioned color extension problem. To solve this problem, we apply Lemma 4.2.2 to U_1 first to extend χ to $\Omega(1/\Delta)$ fraction of it, and keep applying this lemma to the remaining uncolored edges of U_1 until they are all colored. We then move to U_2 in the same exact way and extend χ to its edges as well, obtaining a Δ-coloring of the entire G as a result.

The correctness of the algorithm follows immediately by induction. The runtime can also be analyzed as follows. When coloring U_1 (or U_2), each invocation of Lemma 4.2.2 reduces the number of uncolored edges in U_1 (or U_2) by a $(1 - \Omega(1/\Delta))$ factor and thus we apply this lemma a total of $O(\Delta \cdot \log n)$ time. Moreover, each application of Lemma 4.2.2 takes $\tilde{O}(m/\Delta)$ time with high probability. Thus, with high probability, it only takes $\tilde{O}(m)$ time to extend the coloring χ to U_1 and U_2 in the color extension problem. Hence, the runtime of the algorithm, with high probability, follows the recurrence $T(m, \Delta) \leq 2T(m/2, \Delta/2) + \tilde{O}(m)$, and thus itself is $\tilde{O}(m)$ time.

Finally, note that with $O(m)$ preprocessing time, we can maintain access to the "auxiliary data structure" throughout the repeated invocations of Lemma 4.2.2 above: All we need to do is to maintain a counter for each color $\gamma \in [\Delta]$, which keeps track of how many edges in G are currently assigned the color γ in χ. We maintain these counters in a balanced search tree, and update the relevant counter whenever we change the color assignment of an edge in G. □

The rest of this section is dedicated to the proof of Lemma 4.2.2.

4.2.1 Our Bipartite Color Extension Algorithm in Lemma 4.2.2

At a high level, our algorithm for Lemma 4.2.2 consists of the following three steps.

1. Pick the two least common colors $\alpha, \beta \in [\Delta]$ in Δ. This implies that there are at most $O(m/\Delta)$ edges in G that are colored with α or β in χ.

2. Modify the coloring χ so that $\Omega(\lambda/\Delta)$ of the edges $(u, v) \in U$ either receive a color under χ, or have $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$. While implementing this step, we ensure that the total number of edges with colors α or β remains at most $O(m/\Delta)$.

3. Let Φ denote the set of edges $(u, v) \in U$ with $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$. We call these edges *popular*. We extend χ to a constant fraction of the edges in Φ, by flipping a set of maximal $\{\alpha, \beta\}$-alternating paths.

We now formalize the algorithm; the pseudocode is provided in Algorithm 4. As input, we are given a bipartite graph $G = (V, E)$, a partial Δ-edge coloring χ of G, and a matching $U \subseteq E$ of uncolored edges of size λ.

The algorithm starts by fixing the two least common colors $\alpha, \beta \in [\Delta]$ in χ. The main part is the **while** loop in Line 3, which runs in *iterations*. In each iteration of the **while** loop, the algorithm samples an edge $e = (u, v)$ from U u.a.r. and *attempts* to either (1) directly extend the coloring χ to e (see Line 10), which adds e to a set $C \subseteq U$ of colored edges in U or (2) modify χ so that $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$—we refer to this as making the edge (u, v) *popular*—, which adds e to a set $\Phi \subseteq U$ of popular edges (see Line 20). The attempt to modifying χ is done by essentially finding a maximal $\{c_u, \alpha\}$-alternating path P_u starting at u and a $\{c_v, \beta\}$-alternating path P_v starting at v for $c_u \in \mathsf{miss}_\chi(u)$ and $c_v \in \mathsf{miss}_\chi(v)$ (see Line 12 and Lines 14 and 15). The modification itself is done only if P_u and P_v do not intersect any other popular edge already in Φ.

We say that the concerned iteration of the **while** loop `FAILS` if it chooses an already colored edge in C (Line 6), or modifying the color leads to an already popular edge in Φ to no longer remain popular (Line 17); otherwise, we say the iteration *succeeds*. As stated earlier, the algorithm maintains a subset $\Phi \subseteq U$ of popular edges, and a subset of edges $C \subseteq U$ that got colored since the start of the **while** loop. Initially, we have $C = \Phi = \emptyset$. Thus, the quantity $|\Phi| + |C|$ denotes the number of successful iterations of the **while** loop that have been performed so far. The algorithm performs iterations until $|\Phi| + |C| = \Omega(\lambda/\Delta)$, and then it proceeds to extend the coloring χ to at least a constant fraction of the edges in Φ by finding $\{\alpha, \beta\}$-alternating paths for edges in Φ that admits such paths (see the **for** loop in Line 22).

4.2.2 Analysis of the Bipartite Color Extension Algorithm

We start by summarizing a few key properties of Algorithm 4.

Claim 4.2.3. *Throughout the* **while** *loop in Algorithm 4, there are at most $O(m/\Delta)$ edges in G that receive either the color α or the color β, under χ.*

Proof. We start with $O(m/\Delta)$ such edges in G and each successful iteration of the **while** loop increases the number of such edges by $O(1)$, and there are $O(\lambda/\Delta) = O(m/\Delta)$ such iterations. □

Lemma 4.2.4. *Throughout the execution of the* **while** *loop in Algorithm 4, the following conditions hold: (i) the set C consists of all the edges in U that are colored under χ; (ii) for every edge $(u, v) \in \Phi$, we have $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$.*

Proof. Part (i) of the lemma follows from Line 2 and Line 10. For part (ii), consider an edge $e = (u, v)$ that gets added to Φ. This happens only after flipping the paths P_u and P_v in Line 19. Just before we execute Line 19, the following conditions hold:

- $\alpha \neq \beta$ (see Line 1).
- $c_u \in \mathsf{miss}_\chi(u)$ and $c_v \in \mathsf{miss}_\chi(v)$ (see Line 8).
- $c_u \neq c_v$, $c_u \neq \beta$ and $c_v \neq \alpha$ (see Line 12).
- If $\alpha \in \mathsf{miss}_\chi(u)$ then $P_u = \emptyset$ (see Line 14), and if $\beta \in \mathsf{miss}_\chi(v)$ then $P_v = \emptyset$ (see Line 15).

Algorithm 4: BipartiteExtension(G, χ, U)

1 Let $\alpha, \beta \in [\Delta]$ be the two least common colors in χ // We have $\alpha \neq \beta$
2 Initialize $\Phi \leftarrow \emptyset$, $C \leftarrow \emptyset$, and set $\lambda \leftarrow |U|$
3 **while** $|\Phi| + |C| < \lambda/(10\Delta)$ **do**
4 Sample an edge $e = (u, v) \sim U$ independently and u.a.r.
5 **if** $(u, v) \in \Phi \cup C$ **then**
6 The iteration FAILS
7 **go to Line 3**
8 Identify (arbitrarily) two colors $c_u \in \mathsf{miss}_\chi(u)$ and $c_v \in \mathsf{miss}_\chi(v)$
9 **if** $c_u = c_v$ **then**
10 Set $\chi(u, v) \leftarrow c_u$ and $C \leftarrow C \cup \{(u, v)\}$
11 **go to Line 3**
12 **if** $c_u = \beta$ *or* $c_v = \alpha$ **then**
13 Set $(u, v) \leftarrow (v, u)$ // Now $c_u \neq c_v$, $c_u \neq \beta$, $c_v \neq \alpha$ (see Lines 9 and 12)
14 Let P_u be the maximal $\{c_u, \alpha\}$-alternating path starting at u ($P_u = \emptyset$ if $\alpha \in \mathsf{miss}_\chi(u)$)
15 Let P_v be the maximal $\{c_v, \beta\}$-alternating path starting at v ($P_v = \emptyset$ if $\beta \in \mathsf{miss}_\chi(v)$)
16 **if** *either* P_u *or* P_v *ends at a node that is incident on some edge in* $\Phi \setminus \{e\}$ **then**
17 The iteration FAILS
18 **else**
19 Modify χ by flipping the alternating paths P_u and P_v
20 Set $\Phi \leftarrow \Phi \cup \{(u, v)\}$ // Now $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$
21 $\Phi' \leftarrow \Phi$
22 **for** *each edge* $e = (u, v) \in \Phi'$ **do**
23 $\Phi' \leftarrow \Phi' \setminus \{e\}$
24 W.l.o.g., suppose that $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$
25 **if** *there exists a color* $c \in \{\alpha, \beta\}$ *such that* $c \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$ **then**
26 $\chi(u, v) \leftarrow c$
27 **else**
28 Let $P_u^\star$ be the maximal $\{\alpha, \beta\}$-alternating path starting at u
29 (Since G is bipartite, $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$, $P_u^\star$ does *not* end at v)
30 Modify χ by flipping the alternating path $P_u^\star$, and set $\chi(u, v) \leftarrow \beta$
31 **if** *the path* $P_u^\star$ *ends at a node that is incident on some edge in* $e' \in \Phi' \setminus \{e\}$ **then**
32 $\Phi' \leftarrow \Phi' \setminus \{e'\}$

- The path P_u (resp. P_v) does *not* end at that a vertex that is incident on some edge in $\Phi \setminus \{e\}$ (see Lines 16 and 17), although it might possibly end at v (resp. u).

From these conditions, it follows that the paths P_u and P_v are edge-disjoint, and after we flip them in Line 19, we have $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$ in Line 20.

In subsequent iterations of the **while** loop, the only places where we change the coloring χ are Lines 10 and 19. Since the edges in U form a matching, changing the coloring in Line 10 cannot affect whether or not the edge $(u, v) \in \Phi$ remains popular (i.e., has $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$). Finally, during a subsequent iteration of the **while** loop where we sample an edge $(u', v') \sim U$, we flip the paths $P_{u'}, P_{v'}$ in Line 19 only if their endpoints are *not* incident on any edges in $\Phi \setminus \{(u', v')\}$ (see Line 16), and in particular, on u or v. Thus, this operation cannot change what colors are available at u and v, and so cannot change whether or not the edge $(u, v) \in \Phi$ remains popular. □

We use the following lemma to bound the number of iterations of the **while** loop in Algorithm 4.

Lemma 4.2.5. *Each iteration of the* **while** *loop in Algorithm 4 increases the value of* $|\Phi| + |C|$ *by an additive one, with probability at least* $1/2$ *and otherwise keep it unchanged.*

Proof. Fix any given iteration of the **while** loop. At the start of this iteration, we sample an edge from U u.a.r. We say that an edge $e \in U$ is *bad* if the iteration `FAILS` when we sample e (see Line 6 and Line 17), and *good* otherwise. Note that if we sample a good edge $e \in U$, then the iteration either adds one edge to the set Φ (see Line 20), or adds one edge to the set C (see Line 10). In other words, if we sample a good (resp. bad) edge $e \in U$ at the start of the iteration, then this increases in the value of $|\Phi| + |C|$ by one (resp. keeps the value of $|\Phi| + |C|$ unchanged). We will show that at most $\lambda/2$ edges in U are bad. Since $|U| = \lambda$, this will imply the lemma.

To see why this claimed upper bound on the number of bad edges holds, first note that there are $(|\Phi| + |C|)$ many bad edges that cause the iteration to `FAIL` in Line 6. It now remains to bound the number of bad edges which cause the iteration to `FAIL` in Line 17.

Towards this end, note that for each edge $(u', v') \in \Phi$, there are at most 4Δ many maximal $\{\alpha, \cdot\}$- or $\{\beta, \cdot\}$-alternating paths that end at either u' or v'. Furthermore, each such alternating path has its other endpoint incident on at most one edge in U since the edges in U form a matching. Thus, for each edge $(u', v') \in \Phi$, there are at most 4Δ many edges $f_{(u',v')} \in U$ that satisfy the following condition: Some alternating path constructed by the algorithm after sampling $f_{(u',v')}$ ends at either u' or v' (see Line 14 and Line 15). Each such edge $f_{(u',v')}$ is a bad edge which causes the iteration to `FAIL` in Line 17, and moreover, only such edges are causing the iteration to `FAIL` in Line 17. Thus, the number of such bad edges is at most $|\Phi| \cdot 4\Delta$.

To summarize, the total number of bad edges is at most $(|\Phi| + |C|) + |\Phi| \cdot 4\Delta < \lambda/2$, where the last inequality holds since $|\Phi| + |C| < \lambda/(10\Delta)$ (see Line 3). This concludes the proof. □

Similarly, we can bound the expected runtime of each iteration of the **while** loop in Algorithm 4.

Lemma 4.2.6. *Each iteration of the* **while** *loop in Algorithm 4 takes* $\tilde{O}(m/\lambda)$ *time in expectation, regardless of the outcome of previous iterations.*

Proof. Alternating path flips can be done in time proportional to the path lengths using standard data structures, so we only need to analyze the path lengths. Fix any given iteration of the **while** loop. At the start of this iteration, we can classify the edges in U into one of the following three categories: An edge $e \in U$ is of "Type I" if the iteration ends at Line 7 when we sample e, is of "Type II" if the iteration ends at Line 11 when we sample e, and is of "Type III" otherwise. Let λ_1, λ_2 and λ_3 respectively denote the total number of Type I, Type II and Type III edges, with $\lambda_1 + \lambda_2 + \lambda_3 = \lambda$. For every Type III edge $e = (u, v) \in U$, we refer to the alternating paths P_u and P_v (see Line 14 and Line 15) as the "characteristic alternating paths" for e. Let $\mathcal{P}_3$ denote the collection of characteristic alternating paths of all Type III edges. Since the set of Type III edges is a subset of U, they form a matching, and hence different paths in $\mathcal{P}_3$ have different starting points. Furthermore, every path in $\mathcal{P}_3$ is either a maximal $\{\alpha, \cdot\}$-alternating path or a maximal $\{\beta, \cdot\}$-alternating path. Accordingly, Claim 4.2.3 implies that the total length of all the paths in $\mathcal{P}_3$ is at most $O((m/\Delta) \cdot \Delta) = O(m)$.

Now, if at the start of the iteration, we happen to sample either a Type I or a Type II edge $e \in U$, then the concerned iteration takes $O(1)$ time. In the paragraph below, we condition on the event that the edge $e = (u, v) \in U$ sampled at the start of the iteration is of Type III.

Using appropriate data structures, the time taken to implement the concerned iteration is proportional (up to $\tilde{O}(1)$ factors) to the lengths of the alternating paths P_u and P_v (see Line 14 and Line 15). The key observation is that for each $x \in \{u, v\}$, the path P_x is sampled almost uniformly (i.e., with probability $\Theta(1/\lambda_3)$) from the collection $\mathcal{P}_3$. Since the total length of all the paths in $\mathcal{P}_3$ is $O(m)$, it follows that the expected length of each of the paths P_u, P_v is $O(m/\lambda_3)$.

To summarize, with probability λ_3/λ, we sample a Type III edge at the start of the concerned iteration of the **while** loop, and conditioned on this event the expected time spent on that iteration is $\tilde{O}(m/\lambda_3)$. In contrast, if we sample a Type I or a Type II edge at the start of the concerned iteration, then the time spent on that iteration is $O(1)$. This implies that we spend at most $\tilde{O}(m/\lambda_3) \cdot (\lambda_3/\lambda) + O(1) = \tilde{O}(m/\lambda)$ expected time per iteration of the **while** loop. □

Finally, we show that in the very last step of the algorithm, the **for** loop in Line 22, the algorithm succeeds in coloring a constant fraction of popular edges.

Lemma 4.2.7. *The* **for** *loop in Line 22 extends the coloring χ to at least half of the edges in Φ.*

Proof. Consider any given iteration of the **for** loop where we pick an edge $e = (u, v) \in \Phi'$ in Line 22, where w.l.o.g. $\alpha \in \mathsf{miss}_\chi(u)$ and $\beta \in \mathsf{miss}_\chi(v)$. It is easy to verify that during this iteration, we successfully extend the coloring χ to e, either in Line 26 or in Line 30. In the latter case, we crucially rely on the fact that the graph G is bipartite (see Line 29), and hence the maximal $\{\alpha, \beta\}$-alternating path $P_u^\star$ starting at u cannot end at v; in fact, this is the only place where we rely on the biparteness of G. Lines 31 and 32 ensure that the following invariant is satisfied: For every edge $e' = (u', v') \in \Phi'$, we have $\alpha \in \mathsf{miss}_\chi(u')$ and $\beta \in \mathsf{miss}_\chi(v')$, i.e., the edge e' is popular; indeed, Lemma 4.2.4 implies that this invariant holds just before the **for** loop starts (see Line 21), and any

edge $e' \in \Phi'$ that may violate this invariant at a later stage, which may only occur due to flipping an alternating path that ends at a node incident on e' (in Line 31), is removed from Φ' (in Line 32).

Now, the lemma follows from the observation that each time we successfully extend the coloring to one edge e in Φ', we remove at most one other edge $e' \neq e$ from Φ' (see Line 32), due to the vertex-disjointness of the edges in $U \supseteq \Phi \supseteq \Phi'$. □

We are now ready to conclude the running time analysis of Algorithm 4 and establish the required lower bound on the number of newly colored edges in U under χ by this algorithm.

Lemma 4.2.8. *Algorithm 4 takes $\tilde{O}(m/\Delta)$ time in expectation and extends χ to $\Omega(\lambda/\Delta)$ new edges.*

Proof. We start with the runtime analysis:

- Line 1 can be implemented in $\tilde{O}(1)$ time using the auxiliary data structure, and Lines 2 and 21 take constant time.
- Next, we bound the running time of the **while** loop (Line 3). For any integer $k \geq 0$, let $T(k)$ denote the expected runtime of **while** loop if we start the loop under the condition that $|\Phi| + |C| = k$. We are interested in $T(0)$ and we know that $T(\lambda/10\Delta) = O(1)$ by the termination condition of the loop. By Lemma 4.2.5 and Lemma 4.2.6, for any $0 < k < \lambda/10\Delta$, we have,

$$T(k) \leq \tilde{O}(m/\lambda) + \frac{1}{2} \cdot T(k) + \frac{1}{2} \cdot T(k+1),$$

  where we additionally used the monotonicity of $T(\cdot)$, as well as the fact that each while-loop of Algorithm 4 has expected runtime $\tilde{O}(m/\lambda)$ regardless of previous iterations, according to Lemma 4.2.6. Thus, $T(k) \leq T(k+1) + \tilde{O}(m/\lambda)$ and hence $T(0) \leq \lambda/10\Delta \cdot \tilde{O}(m/\lambda) = \tilde{O}(m/\Delta)$.
- Finally, since the total number of edges with colors α and β just before Line 22 is $O(m/\Delta)$ (see Claim 4.2.3), the **for** loop can be implemented in $\tilde{O}(m/\Delta)$ time deterministically in a straightforward manner.

Thus, the total runtime is $\tilde{O}(m/\Delta)$ in expectation.

We now establish the bound on the number of newly colored edges. When the **while** loop terminates, we have $|\Phi| + |C| \geq \lambda/(10\Delta)$ (see Line 3), and all the edges in C are colored under χ (see Lemma 4.2.4). Next, by Lemma 4.2.7, the **for** loop in Line 22 further extends the coloring χ to a constant fraction of the edges in Φ, by only flipping $\{\alpha, \beta\}$-alternating paths. Consequently, we get at least $\Omega(\lambda/\Delta)$ newly colored edges in U under χ. This concludes the proof. □

We can now conclude the proof of Lemma 4.2.2. To achieve the algorithm in this lemma, we simply run Algorithm 4 in parallel $\Theta(\log n)$ times and use the coloring of whichever one finishes first (and terminate the rest at that point). This ensures the high probability guarantee of Lemma 4.2.2 still in $\tilde{O}(m/\Delta)$ runtime. This concludes the entire proof.

4.2.3 Extension to General Graphs: Roadmap for Chapter 7

In our Lemma 4.2.2, we crucially need the graph G to be bipartite while executing Lines 28 to 30 in Algorithm 4. Otherwise, if G contains odd cycles, then the maximal $\{\alpha, \beta\}$-alternating path $P_u^\star$ starting from u can end at v. In that case, the color β will no longer be missing at v once we flip the path $P_v^\star$, and so we will not be able to extend the coloring χ to the edge (u, v) via $\chi(u, v) \leftarrow \beta$. We shall emphasize that this is not a minor technicality, but rather the key reason general graphs are not necessarily Δ edge colorable and rather require $(\Delta + 1)$ colors.

The standard machinery to deal with this issue is the Vizing fan (see Section 3.2). However, if we try to use Vizing fans inside the framework of Algorithm 4 in a naive manner, then we lose control over one of the colors in the alternating path being flipped while extending the coloring to an edge, leading to a weaker averaging argument and a running time of $\tilde{O}(\Delta m)$ instead of $\tilde{O}(m)$.

To address this bottleneck, one of our key conceptual contributions is to focus on Vizing fans with respect to an object called a *separable collection of u-components* (see Section 7.1). Using this concept, in Section 7.2 we present our algorithmic framework in general graphs. Our main result (see Theorem 7.2.1) relies upon two fundamental subroutines. The second subroutine (see Lemma 7.2.3) generalizes Algorithm 4 presented in this section. In contrast, the first subroutine (see Lemma 7.2.2) either efficiently extends the current coloring to a constant fraction of the uncolored edges, or changes the colors of some edges in the input graph so as to create a situation whereby we can invoke Lemma 7.2.3. We devote Sections 7.3 and 7.4 towards proving Lemmas 7.2.2 and 7.2.3, respectively. For clarity of presentation, we defer the details on supporting data structures to Section 7.5, such as the use of hash tables or binary search tree data structures for basic operations like picking a missing color at a vertex. Finally, Section 7.6 shows how to optimize our algorithmic framework further and achieve an $O(m \log n)$ time with high probability.

4.3 Vizing's Theorem in Deterministic Almost-Linear Time

In this section, we provide an overview of our deterministic almost-linear time algorithm for $(\Delta+1)$-coloring. We summarize this algorithm in Theorem 1.1.2, which we restate below. We give the full details of this algorithm Chapter 8.

Theorem 1.1.2. *There is a deterministic algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta + 1)$-coloring of G in $m \cdot 2^{O(\sqrt{\log \Delta})} \cdot \log n = m^{1+o(1)}$ time.*

All recent randomized edge coloring algorithms that break the $\tilde{O}(m\sqrt{n})$ bound rely crucially on some form of a *sublinear time* algorithm or subroutine. For example, the algorithms in [BCC+24, BCSZ25], as well as our algorithm from Section 4.1, use randomization to find one or many color-alternating paths in $\tilde{O}(n)$ time, and the algorithm of [Ass25] iteratively finds large matchings in $\tilde{O}(n)$ time. While sublinear time algorithms are a very powerful tool for designing randomized algorithms, they are almost always impossible to de-randomize.

Our algorithm in Theorem 1.1.2 builds on our framework from Section 4.1, derandomizing the main "color type reduction" technique, by crucially replacing the sublinear time subroutine with a new "gradual sparsification approach". At a high level, this algorithm is considerably slower and runs in almost-linear time—compared to the sublinear (namely, $O(m/\Delta)$) time guarantee of its counterpart in Section 4.1—but can instead apply the required "type reduction" to an almost constant fraction of the graph—as opposed to only a $\Theta(1/\Delta)$ fraction.

4.3.1 Overview of Our Deterministic Algorithm

To highlight the main ideas behind our approach, in this section we instantiate our algorithm on bipartite graphs. In the rest of this section, we present an overview of the proof of the theorem below; in the runtime bound we will not optimize the logarithmic factors.

Theorem 4.3.1. *There is a deterministic algorithm that, given as input a* bipartite *graph $G = (V, E)$ with n vertices, m edges and maximum degree Δ, and a partial coloring $\chi : E \to [\Delta] \cup \{\perp\}$ such that the set $U = \{e \in E : \chi(e) = \perp\}$ of uncolored edges forms a matching, extends χ to the edges in U in time $\tilde{O}\left(m \cdot 2^{\Theta(\sqrt{\log \Delta})}\right)$.*

We note that using a standard *Euler partitioning* technique, Theorem 4.3.1 immediately implies an $\tilde{O}\left(m \cdot 2^{\Theta(\sqrt{\log \Delta})}\right)$ time algorithm for Δ-coloring a bipartite graph; see the last two paragraphs of Section 8.2.1 for a detailed discussion on this technique. Moreover, in this technical overview section, we ignore low-level implementation details of the supporting data structures, and hide polylogarithmic factors in runtime inside the $\tilde{O}(\cdot)$ notation. Finally, for simplicity of exposition, we assume that the input graph G is almost-regular, as described below.

Assumption 4.3.2. *In the input graph $G = (V, E)$, every vertex $v \in V$ has degree at least $\Omega(\Delta)$. Thus, as a corollary, we have $m = \Theta(n\Delta)$.*

Before proceeding with the proof-sketch of Theorem 4.3.1, we make two important remarks. First, the algorithmic framework developed in this section almost seamlessly extends to $(\Delta + 1)$-coloring in general graphs, via the machinery of *u-fans* (see Section 8.1.1). Thus, although Theorem 4.3.1 in itself does not give us any new result, for it was known since the 1980s how to compute a Δ-coloring in a bipartite graph in deterministic near-linear time [CH82], *our approach* for deriving Theorem 4.3.1 contains all the key conceptual insights that eventually lead to Theorem 1.1.2. Second, we use Assumption 4.3.2 purely for ease of exposition in this technical overview; this assumption does not take anything away from the key insights underpinning our approach.

Roadmap. In Section 4.3.2, we introduce some key notations and concepts. Section 4.3.3 presents Theorem 4.3.3, which summarizes a *type sparsification* framework that underpins our entire algorithm. Immediately after stating Theorem 4.3.3, we explain how it implies Theorem 4.3.1. In Section 4.3.4, we present a *randomized* algorithm for type sparsification. In Section 4.3.5, we compare this new randomized algorithm against our algorithm from Section 4.1, and point out the

barrier towards efficiently derandomizing the latter. Section 4.3.6 demonstrates how the new randomized algorithm from Section 4.3.4 helps us overcome this barrier. Specifically, we derandomize the algorithm from Section 4.3.4, which leads us to the proof of Theorem 4.3.3.

4.3.2 Notations and Preliminaries

A **type** is an unordered pair of distinct colors from C. For any two subsets $A, B \subseteq C$, we slightly abuse the notation and define $A \times B := \{\{\alpha, \beta\} : \alpha \in A, \beta \in B\}$ to be the set of types with one color in A and the other color in B. Given any type $\tau = \{\alpha, \beta\} \in C \times C$ an **alternating path** P of type τ is a *maximal* path in $G = (V, E)$ whose edges are alternatively colored with α and β. We use the phrase **flipping the alternating path** P to denote an operation where every edge on P with color α (resp. β) changes its color to β (resp. α). It is easy to verify that the underlying partial coloring continues to remain valid even after we flip an alternating path. We say that **an uncolored edge** $(u, v) \in E$ **is of type** $\{\alpha, \beta\}$ **iff** $\alpha \in \mathsf{miss}_\chi(u)$ **and** $\beta \in \mathsf{miss}_\chi(v)$ (or vice versa). We say that an **alternating path (resp. uncolored edge) is of type** $A \times B$ **iff it is of type** τ **for some** $\tau \in A \times B$.

4.3.3 The Framework: Type Sparsification

The theorem below encapsulates the main technical insight used to obtain our deterministic algorithm.

Theorem 4.3.3. *Consider any palette C of colors, and any partial coloring $\chi : E \longrightarrow C \cup \{\perp\}$ of the input graph $G = (V, E)$, such that the set $U = \{e \in E : \chi(e) = \perp\}$ of uncolored edges forms a matching. Define $\lambda = |U|$. Fix any parameter $\eta \in [|C|]$ such that $|C|/(2\eta)$ is an integer, and any partition of the palette C into η subsets $\mathcal{C}_1, \ldots, \mathcal{C}_\eta \subseteq C$, such that $|\mathcal{C}_i| = |C|/\eta$ for all $i \in [\eta]$.*

Then, in $\tilde{O}(m \cdot \mathrm{poly}(\eta))$ time, we can change the colors assigned to some of the edges in $E \setminus U$, and ensure that after these changes a constant fraction of the edges in U have types in $\bigcup_{k=1}^{\eta} (\mathcal{C}_k \times \mathcal{C}_k)$.

Remark. Unless explicitly specified otherwise, throughout the rest of Section 4.3 we will use the symbol $C = \{1, \ldots, \Delta\}$ to denote a palette of Δ colors, where Δ is the maximum degree of the input graph $G = (V, E)$. However, Theorem 4.3.3 holds even if $|C| < \Delta$, as long as $\chi : E \longrightarrow C \cup \{\perp\}$ remains a valid partial coloring.

We say that Theorem 4.3.3 achieves **type sparsification**, for the following reason. Initially, the types of the edges in U are chosen from the set $C \times C = [\Delta] \times [\Delta]$, and there are $O(\Delta^2)$ many such types. However, after we run the deterministic algorithm guaranteed by this theorem, the types of a constant fraction of the edges in U are chosen from $\bigcup_{k=1}^{\eta} (\mathcal{C}_k \times \mathcal{C}_k)$, and there are $\eta \cdot O((\Delta/\eta)^2) = O(\Delta^2/\eta)$ many such types. So, for a constant fraction of the edges in U, the set of possible types shrinks by a factor of $\Theta(\eta)$; we say that such edges in U **survive** this type sparsification.

Intuitively, imagine that we have a $\eta \times \eta$ matrix representing a heatmap, where the (i, j)-th entry is the number of edges in U whose type belongs to $\mathcal{C}_i \times \mathcal{C}_j$. Suppose that the number of edges

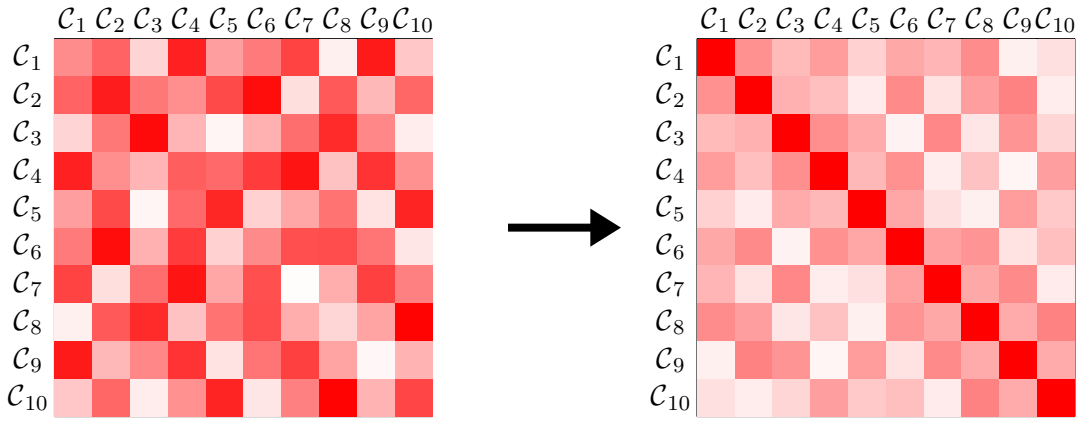

Figure 4.4: In this example, we have $\eta = 10$. The matrix heatmap is initially the left one. After we run the algorithm of Theorem 4.3.3, most weights are concentrated along the diagonal.

with types in each $\mathcal{C}_i \times \mathcal{C}_j$ initially is more or less balanced, so that, for each $1 \leq i, j \leq \eta$, the number of edges in U with types in $\mathcal{C}_i \times \mathcal{C}_j$ is at most $|U|/\eta^2$. Then the heatmap initially looks quite uniform. The goal of Theorem 4.3.3 is to modify the partial coloring χ so that this matrix looks more diagonal. See Figure 4.4 for an illustration.

We now explain why Theorem 4.3.3 implies Theorem 4.3.1. Fix a small $\epsilon \in (0, 1)$. For now, think of ϵ as being a small constant. Set $\eta := \Delta^{\epsilon}$. For simplicity, suppose that Δ (the maximum degree of the input graph) is 2 times an integer power η, i.e. $\Delta = 2 \cdot \eta^q$ for some $q \in \mathbb{N}$.[2]

Next, we apply the type sparsification algorithm from Theorem 4.3.3 on the input graph $G = (V, E)$. This takes $\tilde{O}\left(m \cdot \Delta^{\Theta(\epsilon)}\right)$ time. Once the algorithm finishes execution, let $U^{(s)} \subseteq U$ be the set of surviving uncolored edges. Consider the following natural partition of $U^{(s)}$ into subsets $U_1, \ldots, U_\eta \subseteq U^{(s)}$, where $U_i := \{e \in U^{(s)} : e \text{ is of type } \mathcal{C}_i \times \mathcal{C}_i\}$ for each $i \in [\eta]$. Theorem 4.3.3 guarantees that $|U^{(s)}| \geq \lambda/\kappa$, where $\lambda = |U|$ and $\kappa > 0$ is a sufficiently large absolute constant. Now, for each $i \in [\eta]$, define the subgraph $\mathcal{G}_i = (V, \mathcal{E}_i)$, where $\mathcal{E}_i := U_i \cup \{e \in E : \chi(e) \in \mathcal{C}_i\}$. Let $\chi_i : \mathcal{E}_i \longrightarrow \mathcal{C}_i \cup \{\perp\}$ denote the restriction of χ in $\mathcal{G}_i$, so that we have $\chi_i(e) = \chi(e)$ for all $e \in \mathcal{E}_i$. Note that χ_i is a valid partial coloring with the palette $\mathcal{C}_i$ in $\mathcal{G}_i$, and that $|\mathcal{C}_i| = \Delta/\eta$. Furthermore, the edge-sets $\{\mathcal{E}_i\}_{i \in [\eta]}$ and the palettes $\{\mathcal{C}_i\}_{i \in [\eta]}$ are both mutually disjoint.

We now recursively invoke the type sparsification algorithm from Theorem 4.3.3 on $(\mathcal{G}_i, \chi_i, \mathcal{C}_i)$, for all $i \in [\eta]$. We refer to the quantity $|\mathcal{C}_i|$ as the **palette-size** of the recursive call of $(\mathcal{G}_i, \chi_i, \mathcal{C}_i)$. The base-case of this recursive algorithm occurs when the palette-size becomes small, and in particular 2 (by our assumption that Δ is 2 times an integer power of η);[3] at that point we do not make any further recursive calls. Finally, the parameter η remains equal to Δ^{ϵ}, where Δ is the maximum degree of the initial input graph, *across all the recursive calls.*

It is easy to verify the following facts: (i) Since the palette-size decays by a factor of $\eta = \Delta^{\epsilon}$

[2]We only make this assumption in the technical overview to simplify the technical details, since this leads to a clean recursion tree. We do not need such an assumption in the actual proof.

[3]Without this assumption, the base case occurs when the palette size becomes $O(\eta)$. It is easy to extend the argument of this section to a base case with a palette size of $O(\eta)$, as we demonstrate in the actual proof.

at each recursion layer, the recursion tree of this procedure has depth $1/\epsilon$. (ii) Across any two consecutive layers of this recursion tree, the number of surviving edges in U drops by at most a factor of κ. Thus, across the very last layer of this recursion tree, the total number of surviving uncolored edges is $\Omega\left(\lambda/\kappa^{1/\epsilon}\right)$. (iii) Since the edge-sets and color palettes of different subgraphs at each layer are mutually disjoint, the time spent on each layer of this recursion tree is $\tilde{O}(m\cdot\text{poly}(\eta)) = \tilde{O}\left(m\cdot\Delta^{\Theta(\epsilon)}\right)$. So, the overall runtime across all the layers is $\tilde{O}\left(\epsilon^{-1}\cdot m\cdot\Delta^{\Theta(\epsilon)}\right)$.

Now, consider a "node"[4] $(\mathcal{G}'=(V,\mathcal{E}'),\chi',\mathcal{C}')$ at the last layer of this recursion tree. Let $\mathcal{U}'\subseteq U\cap\mathcal{E}'$ denote the surviving uncolored edges in $\mathcal{G}'$, under χ'. Since $|\mathcal{C}'|=2$, χ' is a valid partial coloring of $\mathcal{G}'$ with palette $\mathcal{C}'$, every edge in $\mathcal{U}'$ is of type $\mathcal{C}'\times\mathcal{C}'$, and the edges in $\mathcal{U}'$ form a matching, it is easy to verify that the subgraph $\mathcal{G}'$ has maximum degree ≤ 2. Thus, in a straightforward manner, we extend the partial coloring χ' to a constant fraction of the edges in $\mathcal{U}'$ in deterministic $\tilde{O}\left(|\mathcal{E}'|\right)$ time by flipping alternating paths of at most one type, without impacting the other "nodes" at the last layer of the recursion tree (as their palettes have no overlap with $\mathcal{C}'$).

To summarize, we infer that in deterministic $\tilde{O}\left(\epsilon^{-1}\cdot m\cdot\Delta^{\Theta(\epsilon)}\right)$ time, the above procedure extends the partial coloring χ to $(1/\kappa^{1/\epsilon})$-fraction of the initial set U of uncolored edges. Repeating this procedure $\tilde{O}\left(\kappa^{1/\epsilon}\right)$ many times, we obtain a deterministic algorithm for extending the initial partial coloring χ to all the edges in U. This algorithm runs in $\tilde{O}\left(\kappa^{1/\epsilon}\cdot\epsilon^{-1}\cdot m\cdot\Delta^{\Theta(\epsilon)}\right)$ time. Now, Theorem 4.3.1 follows if we balance the relevant terms by setting $\epsilon := \Theta(1/\sqrt{\log\Delta})$.

4.3.4 A Randomized Algorithm for Type Sparsification

We now present a *randomized* algorithm that performs the same task as specified in the statement of Theorem 4.3.3. Theorem 4.3.15 summarizes this result. For simplicity of exposition, throughout this section we assume that $C=\{1,\ldots,\Delta\}$ is a palette of Δ colors (see the remark immediately after Theorem 4.3.3). We start by introducing a few more useful notations and terminologies.

For every $k\in[\eta]$, we further partition (arbitrarily) the set $\mathcal{C}_k$ into two equally sized subsets $C_{2k-1}\subseteq\mathcal{C}_k$ and $C_{2k}=\mathcal{C}_k\setminus C_{2k-1}$, so that $|C_{2k-1}|=|C_{2k}|=\Delta/(2\eta)=r$ (say). Thus, the palette $C=[\Delta]$ is partitioned into subsets $C_1,\ldots,C_{2\eta}$, where $\mathcal{C}_k=C_{2k-1}\cup C_{2k}$ for all $k\in[\eta]$. By relabeling the colors, w.l.o.g. we assume that $C_i=[(i-1)r+1,ir]$ for all $i\in[2\eta]$. We say that a type $\tau\in C\times C$ is **uniform** iff $\tau\in C_i\times C_i$ for some $i\in[2\eta]$, and **aligned** iff $\tau\in C_{2k-1}\times C_{2k}$ for some $k\in[\eta]$. A type τ is **social** if it is either uniform or aligned. Finally, we say that an edge $e\in U$ is **social** if e has a type that is social. In words, Theorem 4.3.3 asks us to change the colors of some edges in $E\setminus U$, so as to ensure that a constant fraction of the edges in U become social.

At the start of our type sparsification algorithm, let $U_{\texttt{uni}}\subseteq U$ denote the collection of uncolored edges that have uniform types, and let $U^\star := U\setminus U_{\texttt{uni}}$ denote the remaining set of uncolored edges. By scanning through the edges in U, we can identify the subsets $U_{\texttt{uni}}$ and $U^\star$ in $\tilde{O}(|U|\cdot\Delta)=\tilde{O}(n\Delta)=\tilde{O}(m)$ time (see Assumption 4.3.2). If $|U_{\texttt{uni}}|\geq|U|/2$, then we are done, since a constant fraction of the edges are already social. Thus, for the rest of this section, we assume that $|U^\star|\in[|U|/2,|U|]=[\lambda/2,\lambda]$. Our goal will be to ensure that $\Omega(\lambda)$ many edges in $U^\star$ are of aligned

[4] Not to be confused with a vertex in the input graph.

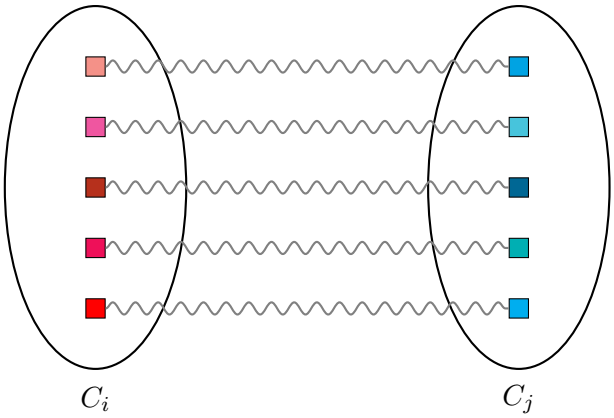

Figure 4.5: The mapping $\phi_{i\to j}$ defines a matching between colors in C_i, C_j. In this example we have drawn 5 colors from color sets C_i, C_j, and colors connected by strings are matched together.

types (by changing the colors of some of the edges in $E \setminus U$).

We will change the colors of some edges in $E \setminus U$, by flipping only some **relevant** alternating paths. To be more specific, for any two distinct indices $i, j \in [2\eta]$, define the mapping $\phi_{i\to j} : C_i \longrightarrow C_j$, as follows. For all $t \in [r]$, we have $\phi_{i\to j}((i-1)r+t) = (j-1)r+t$. In words, $\phi_{i\to j}$ is a one-to-one function which maps the t^{th} color in C_i to the t^{th} color in C_j. Observe that if $\phi_{i\to j}(\alpha) = \beta$, then $\phi_{j\to i}(\beta) = \alpha$; i.e., $\phi^{-1}_{i\to j} = \phi_{j\to i}$. We say that a **type τ is j-relevant, for some index $j \in [2\eta]$ iff $\tau = \{\alpha, \phi_{i\to j}(\alpha)\}$ for some $i \in [2\eta] \setminus \{j\}$ and $\alpha \in C_i$**; we say that **an alternating path P is j-relevant iff it is of a j-relevant type**; for convenience, define Γ_j to be the set of all j-relevant types. Check Figure 4.5 for an illustration.

Observation 4.3.4. *For all $j \in [2\eta]$, there are at most $O(\Delta)$ many types in $C \times C$ that are j-relevant.*

Proof. Suppose that $\tau = (\alpha, \beta)$ is a j-relevant type, and we are told that $\beta \in C_j$. Then, there are at most $(2\eta - 1)$ choices for α. Indeed, once we fix the index $i \in [2\eta] \setminus \{j\}$ such that $\alpha \in C_i$, there is only one unique choice for α which makes the type $\{\alpha, \beta\}$ j-relevant. Hence, there are at most $|C_j| \cdot (2\eta - 1) = O((\Delta/\eta) \cdot \eta) = O(\Delta)$ many j-relevant types. □

Algorithm Description

Our algorithm will consist of η **rounds**. In round $k \in [\eta]$, every alternating path we flip will be either $(2k-1)$-relevant or $(2k)$-relevant. In addition, we will satisfy the invariant stated below.

Invariant 4.3.5. *At the start of round $k \in [\eta]$, we have a collection of mutually-disjoint subsets $U_1, \ldots, U_{k-1} \subseteq U^\star$. For each $k' \in [k-1]$, the following two conditions hold.*

1. *$|U_{k'}| = \lambda/(100\eta)$.*
2. *Every edge $e \in U_{k'}$ is of type $C_{2k'-1} \times C_{2k'}$. Thus, the set $U_{k'}$ consists only of social edges.*

The above invariant guarantees that at the end of the last (i.e., η^{th} round), there are $\eta \cdot \lambda/(100\eta) = \lambda/100$ social edges in $U^\star$, and so the algorithm terminates at this point. It remains to show how to implement a given round. Accordingly, fix any index $k \in [\eta]$. **We will now explain what happens in round** k.

At the start of the round, we have the sets $U_1, \ldots, U_{k-1} \subseteq U^\star$ satisfying Invariant 4.3.5, and we initialize $U_k \leftarrow \emptyset$. Subsequently, we perform a sequence of **iterations**. In a given iteration, we sample an edge $e = (u, v) \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$ uniformly at random. If the iteration **succeeds**, then we manage to ensure that the edge e becomes of type $C_{2k-1} \times C_{2k}$ by changing the colors of some of the edges in $E \setminus U$, and we add the edge e to the set U_k. Otherwise, the iteration **fails**, and the set U_k, along with the colors assigned to the edges in $E \setminus U$, remains unchanged. In addition, we ensure that a successful iteration does not **damage** any edge in $U_1 \cup \cdots \cup U_k$. More formally, during a successful iteration the sets $U_1, \ldots, U_{k-1}$ remain unchanged, and the only change that occurs in U_k is that the edge e gets added to it. Furthermore, for each $k' \in [k]$, every edge $e' \in U_{k'}$ continues to be of type $C_{2k'-1} \times C_{2k'}$. The round terminates after $\lambda/(100\eta)$ successful iterations. It is easy to verify that Invariant 4.3.5 continues to hold at the start of the next round $(k+1)$.

We now focus on explaining how a given iteration works. Let $e = (u, v) \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$ be the edge we sample u.a.r. at the start of the iteration, and suppose that (u, v) is of type $\{\alpha, \beta\} \in C_i \times C_j$, $i, j \in [2\eta]$, $\alpha \in \mathsf{miss}_\chi(u) \cap C_i$, and $\beta \in \mathsf{miss}_\chi(v) \cap C_j$. To highlight the main idea, we focus on the most non-trivial scenario, which occurs when $\{i, j\} \cap \{2k-1, 2k\} = \emptyset$ and $\alpha \neq \beta$. W.l.o.g., suppose that $\alpha < \beta$.[5] We designate the vertex u as being the **left endpoint** of e, and the vertex v as being the **right endpoint** of e. Let $\alpha_{2k-1} := \phi_{i \to (2k-1)}(\alpha)$ and $\beta_{2k} := \phi_{j \to 2k}(\beta)$. Let $P_u(e)$ (resp. $P_v(e)$) be the alternating path starting from u (resp. v) that is of type $\{\alpha, \alpha_{2k-1}\}$ (resp. $\{\beta, \beta_{2k}\}$). Note that the paths $P_u(e)$ and $P_v(e)$ are respectively $(2k-1)$-relevant and $(2k)$-relevant (see Observation 4.3.6). **We will use the notations** $P_u(e)$ **and** $P_v(e)$ **throughout the rest of this section.**

Ideally, we would like to flip the alternating paths $P_u(e)$ and $P_v(e)$: After these flips, we have $\alpha_{2k-1} \in \mathsf{miss}_\chi(u)$ and $\beta_{2k} \in \mathsf{miss}_\chi(v)$. So, the edge (u, v) becomes of type $C_{2k-1} \times C_{2k}$, and we can add the edge (u, v) to the set U_k. There is, however, one serious problem with this strategy. Specifically, it might be the case that the path $P_u(e)$ ends at a vertex u' that is the endpoint of an edge (say) $(u', v') \in U_{k'}$, for some $k' \in [k]$, and if we flip the path $P_u(e)$ then the edge (u', v') will no longer be of type $C_{2k'-1} \times C_{2k'}$. (A similar situation can occur with the path $P_v(e)$.) In other words, flipping the path $P_u(e)$ would *damage* the edge (u', v'). In this case, we say that the edge (u, v) is **bad**, and furthermore, the edge (u', v') is **responsible** for the edge (u, v) being bad. Check Figure 4.6 for an illustration.

Otherwise, if flipping the paths $P_u(e)$ and $P_v(e)$ does not damage any edge in $U_1 \cup \cdots \cup U_k$, then we say that the edge (u, v) is **good**. To summarize, we implement the concerned iteration as follows. We **traverse** the two alternating paths $P_u(e)$ and $P_v(e)$, and confirm whether or not the edge (u, v) is good. If the edge (u, v) is good, then we **flip** the paths $P_u(e)$ and $P_v(e)$, add the edge

[5] Recall that each color is an integer in $[\Delta]$.

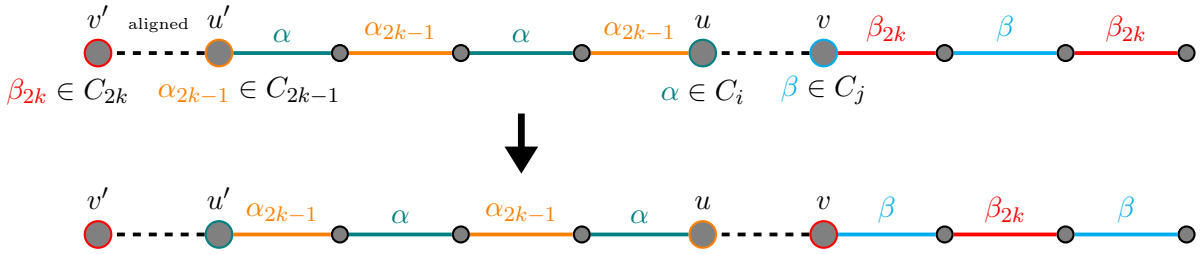

Figure 4.6: In this example, we attempt to align edge (u, v) by flipping the $\{\alpha, \alpha_{2k-1}\}$-alternating path $P_u(e)$ from u and the $\{\beta, \beta_{2k}\}$-alternating path $P_v(e)$ from v. However, flipping the $\{\alpha, \alpha_{2k-1}\}$-alternating path from u damages a previously aligned edge (u', v') as u' will not miss color α_{2k-1} anymore. In this case, (u', v') is responsible for (u, v).

(u, v) to the set U_k, and then proceed towards the next iteration.

Analysis

From the description in Section 4.3.4, it immediately follows that when the algorithm terminates, a constant fraction of the edges in U have types in $\bigcup_{k=1}^{\eta}(\mathcal{C}_k \times \mathcal{C}_k)$, as desired by Theorem 4.3.3. It remains to bound the expected time complexity of this algorithm. Towards this end, we start with a couple of simple observations. Subsequently, Claim 4.3.8 and Corollary 4.3.9 bound the expected runtime of a given iteration within a round $k \in [\eta]$, whereas Claim 4.3.10 and Corollary 4.3.13 bound the probability that a given iteration is successful. Putting everything together, Lemma 4.3.14 bounds the expected runtime of a given round. This, in turn, leads us to Theorem 4.3.15.

Observation 4.3.6. *During any given iteration within a round $k \in [\eta]$, we traverse and/or flip at most two alternating paths $P_u(e)$ and $P_v(e)$, where $e = (u, v) \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$ is the edge we sample at the start of the iteration and u (resp. v) is the left (resp. right) endpoint of e. The paths $P_u(e)$ and $P_v(e)$ are $(2k-1)$-relevant and $(2k)$-relevant, respectively.*

Proof. Follows from the description of the algorithm in Section 4.3.4. □

Observation 4.3.7. *Throughout the duration of round $k \in [\eta]$, we have $|U^\star \setminus (U_1 \cup \cdots \cup U_k)| \in [\lambda/4, \lambda]$.*

Proof. By Invariant 4.3.5, we have $|U_1 \cup \cdots \cup U_k| \leq k \cdot \lambda/(100\eta) \leq \eta \cdot \lambda/(100\eta) = \lambda/100$. The observation follows since we have already assumed that $|U^\star| \in [\lambda/2, \lambda]$, □

Claim 4.3.8. *At any given point in time within a round $k \in [\eta]$, define*

$$\begin{aligned} \mathcal{P}_{\texttt{left}} &:= \{P_{u^\star}(e^\star) : e^\star \in U^\star \setminus (U_1 \cup \cdots \cup U_k), u^\star \text{ is the left endpoint of } e^\star\}, \text{ and} \\ \mathcal{P}_{\texttt{right}} &:= \{P_{v^\star}(e^\star) : e^\star \in U^\star \setminus (U_1 \cup \cdots \cup U_k), v^\star \text{ is the right endpoint of } e^\star\}. \end{aligned}$$

Let $|P|$ denote the length of an alternating path P. Then, we have $\sum_{P \in \mathcal{P}_{\texttt{left}} \cup \mathcal{P}_{\texttt{right}}} |P| = O(m)$.

Proof. Recall that the edge-set $U^\star \setminus (U_1 \cup \cdots \cup U_k) \subseteq U$ forms a matching (see Theorem 4.3.3). By Observation 4.3.4, we have $|\Gamma_{2k-1}| = O(\Delta)$; recall that Γ_{2k-1} is the set of all $(2k-1)$-relevant types. Since the total length of all alternating paths of a given type is at most n, it follows that $\sum_{P \in \mathcal{P}_{\texttt{left}}} |P| \leq |\Gamma_{2k-1}| \cdot n = O(\Delta n) = O(m)$ (see Assumption 4.3.2). Using an analogous argument, we get $\sum_{P \in \mathcal{P}_{\texttt{right}}} |P| = O(m)$. This concludes the proof. □

Corollary 4.3.9. *Each iteration within a round $k \in [\eta]$ takes $\tilde{O}(m/\lambda)$ time in expectation.*

Proof. Let $e = (u, v) \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$ be the edge we sample u.a.r. at the start of the iteration. Using appropriate supporting data structures, it is easy to ensure that the time spent during the given iteration is proportional to the total length of the paths $P_u(e)$ and $P_v(e)$. Thus, from now on, we focus on bounding the expected **length** of each of these two paths.

The edge-set $U^\star \setminus (U_1 \cup \cdots \cup U_k) \subseteq U$ forms a matching (see Theorem 4.3.3). W.l.o.g., we assume that u (resp. v) is the left (resp. right) endpoint of the edge $e = (u, v)$. Accordingly, we have

$$\mathbb{E}\left[|P_u(e)|\right] = \frac{\sum_{P \in \mathcal{P}_{\texttt{left}}} |P|}{|U^\star \setminus (U_1 \cup \cdots \cup U_k)|} \text{ and } \mathbb{E}\left[|P_v(e)|\right] = \frac{\sum_{P \in \mathcal{P}_{\texttt{right}}} |P|}{|U^\star \setminus (U_1 \cup \cdots \cup U_k)|}. \tag{4.1}$$

The corollary now follows from Observation 4.3.7 and Claim 4.3.8. □

Claim 4.3.10. *At the start of each iteration within round $k \in [\eta]$, at least a constant fraction of the edges in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$ are good.*

Proof. The claim holds because of the following two key observations.

Observation 4.3.11. *Each edge $e' \in U_1 \cup \cdots \cup U_{k-1}$ is responsible for at most 4 bad edges.*

Observation 4.3.12. *Each edge $e' \in U_k$ is responsible for at most 4η bad edges.*

Taken together, Observation 4.3.11 and Observation 4.3.12 (along with Invariant 4.3.5) imply that the number of bad edges is at most

$$\sum_{k'=1}^{k-1} 4 \cdot |U_{k'}| + 4\eta \cdot |U_k| \leq 4k \cdot \lambda/(100\eta) + 4\eta \cdot \lambda/(100\eta) \leq 4\eta \cdot \lambda/(100\eta) + \lambda/25 \leq (2/25) \cdot \lambda.$$

Since $|U^\star \setminus (U_1 \cup \cdots \cup U_k)| \in [\lambda/4, \lambda]$ by Observation 4.3.7, *at most* a constant fraction of the edges in $U^\star \setminus (U_1 \cup \ldots \cup U_k)$ are bad; or equivalently, *at least* a constant fraction of the edges in $U^\star \setminus (U_1 \cup \ldots \cup U_k)$ are good. It now remains to explain why the two key observations hold.

For Observation 4.3.11, consider any edge $e' = (u', v') \in U_{k'}$, for some $k' \in [k-1]$. W.l.o.g., let e' be of type $\{\alpha', \beta'\}$, where $\alpha' \in \mathsf{miss}_\chi(u') \cap C_{2k'-1}$ and $\beta' \in \mathsf{miss}_\chi(v') \cap C_{2k'}$ (see Invariant 4.3.5). Now, if e' is responsible for some bad edge $e^\star = (u^\star, v^\star)$, then the following condition must hold. Either there exists an $i \in \{2k-1, 2k\}$ and an $x \in \{u^\star, v^\star\}$ such that the alternating path of type $\{\alpha', \phi_{(2k'-1)\to i}(\alpha')\}$ starting from u' ends at x; or there exists an $i \in \{2k-1, 2k\}$ and an $x \in \{u^\star, v^\star\}$ such that the alternating path of type $\{\beta', \phi_{(2k')\to i}(\beta')\}$ starting from v' ends at x. It is easy to verify that there are at most two such alternating paths that start from u' (resp. v'), one for each

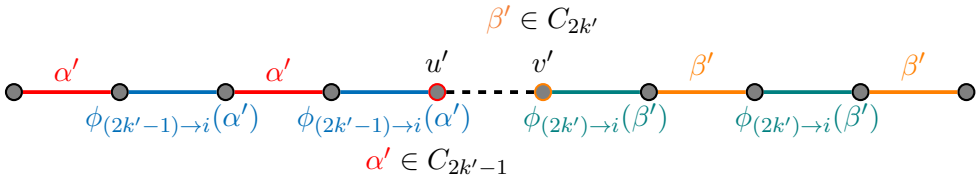

Figure 4.7: In this picture, $e' = (u', v') \in U_{k'}$, for some $k' < k$. We have drawn two alternating paths of types $\{\alpha', \phi_{(2k'-1)\to i}(\alpha')\}$ and $\{\beta', \phi_{(2k')\to i}(\beta')\}$ from u', v' which could damage some $e^\star$.

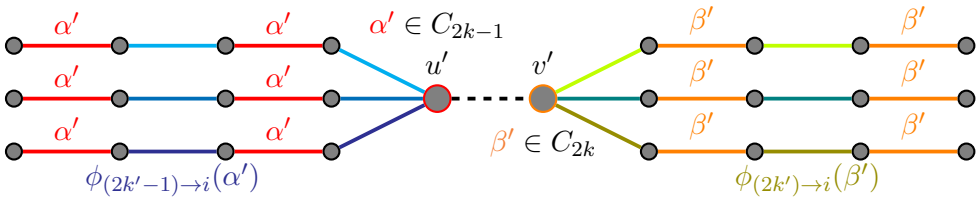

Figure 4.8: In this picture, $e' = (u', v') \in U_k$. We have drawn some alternating paths of types $\{\alpha', \phi_{(2k'-1)\to i}(\alpha')\}$ and $\{\beta', \phi_{(2k')\to i}(\beta')\}$ from u', v' for three choices of $i \in [2\eta] \setminus \{2k-1, 2k\}$.

$i \in \{2k-1, 2k\}$. So, the edge e' can be responsible for at most $2 \times 2 = 4$ bad edges. Check Figure 4.7 for an illustration.

For Observation 4.3.12, consider any edge $e' = (u', v') \in U_k$. W.l.o.g., let e' be of type $\{\alpha', \beta'\}$, where $\alpha' \in \mathsf{miss}_\chi(u') \cap C_{2k-1}$ and $\beta' \in \mathsf{miss}_\chi(v') \cap C_{2k}$ (see Invariant 4.3.5). Now, if e' is responsible for a bad edge $e^\star = (u^\star, v^\star)$, then the following condition must hold. Either there exists an $i \in [2\eta] \setminus \{2k-1, 2k\}$ and an $x \in \{u^\star, v^\star\}$ such that the alternating path of type $\{\alpha', \phi_{(2k-1)\to i}(\alpha')\}$ starting from u' ends at x; or there exists an $i \in [2\eta] \setminus \{2k-1, 2k\}$ and an $x \in \{u^\star, v^\star\}$ such that the alternating path of type $\{\beta', \phi_{(2k)\to i}(\beta')\}$ starting from v' ends at x. It is easy to verify that there are at most $2\eta - 2$ such alternating paths that start from u' (resp. v'), one for each $i \in [2\eta] \setminus \{2k-1, 2k\}$. So, the edge e' can be responsible for at most $2 \cdot (2\eta - 2) \leq 4\eta$ bad edges. Check Figure 4.8 for an illustration. □

Remark. Note that the proofs of Observation 4.3.7, Claim 4.3.8 and Claim 4.3.10 do not rely on randomness in any way; we will use these statements also in the deterministic setting of Section 4.3.6.

Corollary 4.3.13. *Each iteration within a round $k \in [\eta]$ succeeds with constant probability.*

Proof. At the start of the iteration, we sample an edge $e \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$. The iteration succeeds if the sampled edge e is good. The corollary now follows from Claim 4.3.10. □

Lemma 4.3.14. *In the algorithm from Section 4.3.4, each round takes $\tilde{O}(m/\eta)$ expected time.*

Proof. By Corollary 4.3.9, each iteration within a round takes $\tilde{O}(m/\lambda)$ expected time. By Corollary 4.3.13, each iteration succeeds with $O(1)$ probability, hence w.h.p. the number of iterations per round is $\tilde{O}(\lambda/\eta)$. So, the expected time complexity of a round is $\tilde{O}(\lambda/\eta) \cdot \tilde{O}(m/\lambda) = \tilde{O}(m/\eta)$. □

Theorem 4.3.15. *The algorithm from Section 4.3.4 performs the same task as specified in the statement of Theorem 4.3.3, and runs in expected $\tilde{O}(m)$ time.*

Proof. Since the algorithm consists of η rounds, the theorem follows from Lemma 4.3.14. □

4.3.5 Comparison with Our Randomized Algorithm From Section 4.1

We now explain how to derive the result in Section 4.1 from the randomized algorithm presented in Section 4.3.4. Set $C = \{1, \ldots, \Delta\}$ to be the palette of Δ colors, and set $\eta := \Delta/2$. Thus, the palette $C = [\Delta]$ is partitioned into subsets $\mathcal{C}_1, \ldots, \mathcal{C}_\eta \subseteq C$, where $|\mathcal{C}_k| = 2$ for all $k \in [\eta]$. W.l.o.g., suppose that $\mathcal{C}_k = \{2k-1, 2k\}$ for all $k \in [\eta]$.

Next, execute only the *first* round of the algorithm from Section 4.3.4. By Lemma 4.3.14, this takes $\tilde{O}(m/\eta) = \tilde{O}(n\Delta/\eta) = \tilde{O}(n)$ *expected time*. By Invariant 4.3.5, at the end of this round we have a subset $U_1 \subseteq U$ of uncolored edges such that $|U_1| = \Omega(\lambda/\Delta)$, and every edge in U_1 is of type $\{1,2\} \times \{1,2\}$. At this point, our randomized algorithm from Section 4.1 extends the partial coloring χ to a constant fraction of the edges in U_1 in *deterministic* $\tilde{O}(n)$ time, by flipping some alternating paths of type $\{1,2\} \times \{1,2\}$.

To summarize, the above procedure extends the initial partial coloring to $\Omega(1/\Delta)$ fraction of the uncolored edges, in $\tilde{O}(n)$ expected time. Repeating this procedure $\tilde{O}(\Delta)$ times, we get an algorithm that runs in $\tilde{O}(n\Delta) = \tilde{O}(m)$ expected time (see Assumption 4.3.2), and extends the initial partial coloring to *all* the uncolored edges in U. Thus, the only part where randomization is used in this procedure is for this **key task**: Implement round $k \in [\eta]$ of the algorithm from Section 4.3.4 in $\tilde{O}(n)$ time, when $\eta = \Delta/2$. We crucially require that this key task gets implemented deterministically in *sublinear* $\hat{O}(n)$ time[6], if we are to use it to design an almost-linear time deterministic algorithm for Δ-coloring. This requirement is the *only* bottleneck towards our ultimate goal of derandomizing our randomized algorithm from Section 4.1. Nevertheless, the bottleneck seems insurmountable; just like many other computational problems in the sublinear setting, we do not see how to design a deterministic algorithm for this task that does *not* read the entire input graph.

Alternatively, to implement the above randomized algorithm from Section 4.1 in a way that is more compatible with our framework discussed in Section 4.3.4, we could also execute all $\eta = \Delta/2$ rounds of socialization steps before doing any color extensions, and then we would end up with a constant fraction of uncolored edges in U whose types are from $\{1,2\}\times\{1,2\}, \{3,4\}\times\{3,4\}, \ldots, \{\Delta-1, \Delta\} \times \{\Delta-1, \Delta\}$. At this point, we can easily extend the partial coloring to a constant fraction of the edges in U in $\tilde{O}(\eta \cdot n) = \tilde{O}(m)$ time. Alas, derandomizing this alternative approach would basically run into the same problem. The core difficulty in a direct derandomization of the extreme case when $\eta = \Delta/2$ is that the total number of possible types of alternating paths the algorithm could potentially flip is $\Omega(\Delta\eta) = \Omega(\Delta^2)$. Hence, since we are aiming at a total time bound of $\tilde{O}(m) = \tilde{O}(n\Delta)$, the average time spent on each type of alternating paths should be $\tilde{O}(n/\Delta)$. However, in the deterministic setting, for a single type of alternating paths, we cannot find and

[6] $\hat{O}$ hides sub-polynomial factors.

process a $1/\Delta$-fraction of these paths in $\tilde{O}(n/\Delta)$ time, since we can only exploit the property that their total length is bounded by $O(n)$. In our deterministic algorithm, we will set η to be much smaller than $\Delta/2$, namely $\eta = \Delta^{\epsilon}$ for a small ϵ, with the advantage that we will only flip $O(\Delta^{1+\epsilon})$ different types of alternating paths.

Our main conceptual contribution in this section is to formulate the notion of *type sparsification*, as captured in the statement of Theorem 4.3.3, and derive the new randomized algorithm presented in Section 4.3.4. Since Theorem 4.3.3 asks for a time complexity of $\tilde{O}(m \cdot \text{poly}(\eta))$ and the randomized algorithm from Section 4.3.4 has η rounds, this gives us an added flexibility of implementing a given round in $\tilde{O}(m \cdot \text{poly}(\eta))$ time, as opposed to the sublinear $\tilde{O}(m/\eta)$ time guarantee of Lemma 4.3.14. In Section 4.3.6, we show how to exploit this flexibility; and indeed present a *deterministic* $\tilde{O}(m \cdot \text{poly}(\eta))$ time algorithm for implementing a given round. This leads us to Theorem 4.3.3, which, as explained in Section 4.3.3, implies Theorem 4.3.1.

4.3.6 Derandomization: Proof of Theorem 4.3.3

We now show how to derandomize the algorithm from Section 4.3.4. Recall the description of the randomized algorithm from Section 4.3.4. Specifically, all we need is a deterministic subroutine for the following task: Implement a given round $k \in [\eta]$. We devote the rest of this section towards explaining how to perform this task in deterministic $\tilde{O}(m \cdot \text{poly}(\eta))$ time. Since the algorithm from Section 4.3.4 consists of η rounds, this leads us to Theorem 4.3.3.

At any stage during a round $k \in [\eta]$, we say that a set $B \subseteq U^{\star} \setminus (U_1 \cup \cdots \cup U_k)$ of edges is a **batch** iff there exist two *distinct* indices $i, j \in [2\eta]$ such that every edge $e \in B$ is of type $C_i \times C_j$. The **size** of the batch is given by $|B|$. We say that the batch is **good** iff every edge $e \in B$ is good. Our deterministic implementation of a given round $k \in [\eta]$ is based on three key insights.

The first insight is summarized in Claim 4.3.16, which guarantees that at any point in time during round $k \in [\eta]$, either (i) there exist $\Omega(\lambda)$ uncolored edges that are of uniform types, or (ii) there exists a good batch of $\Omega(\lambda/\eta^2)$ edges. Under case (i), these $\Omega(\lambda)$ uncolored edges of uniform types are already social, and so we simply terminate the algorithm at this point. Thus, from now on we assume the existence of a good batch B of $\Omega(\lambda/\eta^2)$ uncolored edges.

The second insight is summarized in Claim 4.3.17, which shows that we can compute the set of good edges (say) $U^{(g)} \subseteq U^{\star} \setminus (U_1 \cup \cdots \cup U_k)$ in deterministic $\tilde{O}(m)$ time. Once we have computed the set $U^{(g)}$, we explicitly scan through the edges in $U^{(g)}$ to identify the largest good batch $B \subseteq U^{(g)}$. From the preceding discussion, we are guaranteed that $|B| = \Omega(\lambda/\eta^2)$. So, we have managed to identify a good batch B of $\Omega(\lambda/\eta^2)$ edges in deterministic $\tilde{O}(m)$ time.

To appreciate the third insight, consider any edge $e = (u, v) \in B$. Since B is a good batch, the edge e is good and has a type that is *not* aligned. Accordingly, if we flip the two alternating paths $P_u(e)$ and $P_v(e)$, then the edge e would become of type $C_{2k-1} \times C_{2k}$ and get added to the set U_k, without damaging any existing edge in $U_1 \cup \cdots \cup U_k$. Let us refer to $P_u(e)$ and $P_v(e)$ as the **characteristic alternating paths** of the edge e. We also say that $P_u(e)$ and $P_v(e)$ are **characteristic alternating paths of the batch** B (since $e \in B$). Now, Claim 4.3.18 and

Claim 4.3.19 allow us to conclude that the types of any two characteristic alternating paths of B are either equal or mutually disjoint. Since the edges in $B \subseteq U$ form a matching (see Theorem 4.3.3), this has the following implications: We can *simultaneously*[7] flip all the characteristic alternating paths of B, without damaging any existing edge in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$. Once we flip these paths, *all* the edges in B get added to the set U_k. Furthermore, the time taken to perform this operation is proportional (up to polylogarithmic factors) to the total length of the characteristic alternating paths (summed over all the edges in B); this, in turn, is at most $O(m)$ as per Claim 4.3.8.

To summarize, what we have described above is a deterministic procedure that runs in $\tilde{O}(m)$ time, and performs the following task: It adds $\Omega(\lambda/\eta^2)$ edges to the set U_k, without damaging any existing edge in $U_1 \cup \cdots \cup U_k$. Thus, we can implement round $k \in [\eta]$ by repeating this procedure $O(\eta)$ times (see Invariant 4.3.5). In other words, there is a deterministic procedure for implementing a given round in $\tilde{O}(m \cdot \text{poly}(\eta))$ time. This implies Theorem 4.3.3, since the algorithm from Section 4.3.4 consists of η rounds.

Claim 4.3.16. *At any stage during a round $k \in [\eta]$, let $\hat{U}_{\texttt{uni}}$ denote the set of edges in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$ that are of uniform types. Then, either $|\hat{U}_{\texttt{uni}}| = \Omega(\lambda)$, or there exists a good batch $B \subseteq U^\star \setminus (U_1 \cup \cdots \cup U_k)$ consisting of $\Omega(\lambda/\eta^2)$ edges.*

Proof. Fix a sufficiently large absolute constant $\kappa > 1$. If $|\hat{U}_{\texttt{uni}}| \geq \lambda/\kappa$, then the claim clearly holds. For the rest of the proof, we assume that $|\hat{U}_{\texttt{uni}}| < \lambda/\kappa$. By Observation 4.3.7 and Claim 4.3.10, there are $\Omega(\lambda)$ good edges in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$. Since κ is a sufficiently large constant and $|\hat{U}_{\texttt{uni}}| < \lambda/\kappa$, there exists a set $S \subseteq U^\star \setminus (U_1 \cup \cdots \cup U_k)$ of $\Omega(\lambda)$ good edges that are *not* uniform. This set S is partitioned into a collection of good batches. The total number of such batches is at most $O(\eta^2)$, since each batch is defined by two distinct indices $i, j \in [2\eta]$. Hence, by a simple averaging argument, there must exist a good batch B of size $\Omega(\lambda/\eta^2)$. □

Claim 4.3.17. *At any stage during a round $k \in [\eta]$, we can compute the set of good edges in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$ in deterministic $\tilde{O}(m)$ time.*

Proof. We scan through all the edges in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$, and for each such edge, we check whether it is good or bad. Consider any edge $e = (u, v) \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$, and let u (resp. v) be its left (resp. right) endpoint. To check whether e is good or bad, all we need to do is traverse the two alternating paths $P_u(e)$ and $P_v(e)$, and confirm whether flipping any of these paths damages some edge in $U^\star \setminus (U_1 \cup \cdots \cup U_k)$. If we use appropriate supporting data structures, then the time taken to implement this step is dominated by the total lengths of the alternating paths $P_u(e)$ and $P_v(e)$. Thus, upto a polylogarithmic factor, the time complexity of the entire procedure is proportional to $\sum_{P \in \mathcal{P}_{\texttt{left}} \cup \mathcal{P}_{\texttt{right}}} |P| = O(m)$, as per Claim 4.3.8. □

Claim 4.3.18. *At any stage during a round $k \in [\eta]$, let $B \subseteq U^\star \setminus (U_1 \cup \cdots \cup U_k)$ be a batch of edges, such that every edge in B is of type $C_i \times C_j$, for $i, j \in [2\eta]$, $i \neq j$. Consider any two distinct edges*

[7] When we say that we can flip these paths *simultaneously*, we mean that we can flip them in any arbitrary order and still have the same effect on the coloring.

$e = (u, v) \in B$ and $e' = (u', v') \in B$, and any two vertices $x \in \{u, v\}$ and $x' \in \{u', v'\}$. Let τ and τ' respectively denote the types of the alternating paths $P_x(e)$ and $P_{x'}(e')$.

Then, it must necessarily be the case that either $\tau = \tau'$ or $\tau \cap \tau' = \emptyset$.

Proof. Let $\{\alpha, \beta\}$ be the type of the edge $e = (u, v)$, with $\alpha \in \mathsf{miss}_\chi(u) \cap C_i$ and $\beta \in \mathsf{miss}_\chi(v) \cap C_j$. Similarly, let $\{\alpha', \beta'\}$ be the type of the edge $e' = (u', v')$, with $\alpha' \in \mathsf{miss}_\chi(u') \cap C_i$ and $\beta' \in \mathsf{miss}_\chi(v') \cap C_j$. W.l.o.g., suppose that $i < j$, i.e., u and u' (resp. v and v') are the left (resp. right) endpoints of e and e. As in Section 4.3.4, we focus on the most non-trivial scenario, where $\{i, j\} \cap \{2k-1, 2k\} = \emptyset$. Now, consider four possible cases.

Case 1: $x = u$ and $x' = u'$. Here, both the types τ and τ' are $(2k-1)$-relevant. Specifically, we have $\tau = \{\alpha, \phi_{i \to (2k-1)}(\alpha)\}$ and $\tau' = \{\alpha', \phi_{i \to (2k-1)}(\alpha')\}$. Since $\phi_{i \to (2k-1)} : C_i \longrightarrow C_{2k-1}$ is a one-to-one mapping, we conclude that either $\tau = \tau'$ or $\tau \cap \tau' = \emptyset$.

Case 2: $x = v$ and $x' = v'$. Here, both the types τ and τ' are $(2k)$-relevant, and the argument is analogous to the one in Case 1.

Case 3: $x = u$ and $x' = v'$. Here, the type τ is $(2k-1)$-relevant and the type τ' is $(2k)$-relevant. Specifically, we have $\tau \in C_i \times C_{2k-1}$ and $\tau' \in C_j \times C_{2k}$. Since $i \neq j$, we conclude that $\tau \cap \tau' = \emptyset$.

Case 4: $x = v$ and $x' = u'$. Here, the type τ is $(2k)$-relevant and the type τ' is $(2k-1)$-relevant, and the argument is analogous to the one in Case 3.

To summarize, from the above four cases, we infer that either $\tau = \tau'$ or $\tau \cap \tau' = \emptyset$. □

Claim 4.3.19. *At any stage during a round $k \in [\eta]$, consider an edge $e = (u, v) \in U^\star \setminus (U_1 \cup \cdots \cup U_k)$ whose type is* not *aligned. Let τ and τ' respectively denote the types of the alternating paths $P_u(e)$ and $P_v(e)$. Then, it must necessarily be the case that $\tau \cap \tau' = \emptyset$.*

Proof. Let $\{\alpha, \beta\}$ be the type of the edge $e = (u, v)$, where $\alpha \in \mathsf{miss}_\chi(u) \cap C_i$, $\beta \in \mathsf{miss}_\chi(v) \cap C_j$, and $i \neq j$. W.l.o.g., suppose that $i < j$. As in Section 4.3.4, we consider the most nontrivial scenario, where $\{i, j\} \cap \{2k-1, 2k\} = \emptyset$. Now, it is easy to verify that $\tau \in C_i \times C_{2k-1}$ and $\tau' \in C_j \times C_{2k}$. Since $i \neq j$, we get $\tau \cap \tau' = \emptyset$. □

Chapter 5

Our Algorithms for Dynamic Edge Coloring

In this chapter, we provide a detailed technical overview of our dynamic algorithms for edge coloring, highlighting our key ideas and techniques. In Section 5.1, we overview our dynamic $(\Delta+1)$-coloring algorithm based on multi-step Vizing chains. In Section 5.2, we instantiate our multi-step Vizing chain algorithm on bipartite graphs to showcase some of our key insights. In Section 5.3, we overview our techniques for designing $(1+\epsilon)\Delta$-coloring algorithms based on the Nibble method. In Sections 5.4 and 5.5, we give detailed technical overviews of our static and dynamic algorithms based on the Nibble method, respectively.

Notation. Sections 5.1 and 5.2 in this chapter use the basic notation from Section 3.1, as well as introducing additional notation. Sections 5.3 to 5.5 use different techniques from the rest of our results, and thus introduce their own notation and are self-contained.

5.1 Dynamic Edge Coloring via Multi-Step Vizing Chains

In this section, we provide an overview our dynamic $(\Delta+1)$-coloring algorithm based on multi-step Vizing chains. We summarize this algorithm in Theorem 1.1.6, which we restate below. We give the full details of this algorithm in Chapter 9.

Theorem 1.1.6. *There is an algorithm that, given a dynamic graph G with maximum degree at most Δ, maintains a $(\Delta+1)$-coloring of G with $\tilde{O}(\Delta^2+\sqrt{\Delta n})$ worst-case update time, against an adaptive adversary, with high probability.*

Our proof of this theorem builds on a sequence of earlier works [DHZ19, Ber22, Chr23]. Our key technical contribution is a subroutine for extending a partial coloring to one more edge within time $\tilde{O}(\Delta^2+\sqrt{\Delta n})$, which immediately yields the dynamic algorithm in Theorem 1.1.6. The best previous time bound of any color extension subroutine is either the trivial $O(n)$, dominated by the length of a Vizing chain, or the bound $\tilde{O}(\Delta^6)$ by Bernshteyn [Ber22], dominated by the length

of a multi-step Vizing chain. There is also a much faster color extension subroutine, by Duan et al. [DHZ19]: it uses $(1+\epsilon)\Delta$ colors within time $\tilde{O}(1/\epsilon^2)$, provided that $\Delta = \tilde{\Omega}(1)$ and $\epsilon = \tilde{\Omega}(1/\sqrt{\Delta})$; we note that the minimum number of colors achievable by [DHZ19] is $\Delta+\tilde{O}(\sqrt{\Delta})$, and the respective runtime is $\tilde{O}(\Delta)$. Even slightly reducing the number of colors below $\Delta + \tilde{O}(\sqrt{\Delta})$, while allowing a higher time of $\tilde{O}(\Delta^2)$, is currently out of reach.

Our color extension subroutine settles for the aforementioned higher running time of $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$; even then, it has to drill much deeper than [DHZ19], since the transition from $\Delta + \tilde{O}(\sqrt{\Delta})$ colors to $\Delta + 1$ has to overcome numerous nontrivial technical hurdles. Ultimately, we manage to produce significantly shorter multi-step Vizing chains than in previous works for $\Delta + 1$ colors, in the regime that Δ is sufficiently large.

5.1.1 Our Techniques

$(\Delta + \eta)$-Color Extension Subroutine. Here, $\eta \geq 1$ is an integer. The input to this subroutine is a graph $G = (V, E)$ with maximum degree Δ, a given edge $e \in E$, and a valid coloring $\chi_{\texttt{init}} : E \setminus \{e\} \to [\Delta + \eta]$ of $G \setminus \{e\}$. The subroutine has to *extend* the *partial* coloring χ to the entire graph G, by assigning some color $c \in [\Delta + \eta]$ to the uncolored edge $e \in E$ and possibly changing the colors of some edges in $E \setminus \{e\}$. Let $\chi_{\texttt{final}} : E \to [\Delta + \eta]$ be the valid $(\Delta + \eta)$-edge coloring of G when the subroutine finishes execution. We define the **cost** incurred by the subroutine to be the number of edges in G that change their colors during this process, i.e., the cost equals $1 + |\{e' \in E \setminus \{e\} : \chi_{\texttt{init}}(e') \neq \chi_{\texttt{final}}(e')\}|$. It is easy to observe that **the runtime of any such subroutine is at least its cost**, because the subroutine needs to spend $\Omega(1)$ time to change the color of any given edge. We now summarize a lower bound derived by Chang et al. [CHL+20].

Theorem 5.1.1 ([CHL+20])**.** *Any $(\Delta+\eta)$-color extension subroutine has cost $\Omega((\Delta/\eta)\log(\eta n/\Delta))$.*

A Major Challenge. In recent years, an influential line of work [DHZ19, SV19, CHL+20, GP20, Ber22, Chr23] has addressed the question of finding a *small augmenting subgraph* to extend a partial coloring to an uncolored edge, primarily from a different vantage point of distributed algorithms. Many of these results were obtained using *multi-step Vizing chains*, a generalization of the central object used in Vizing's original proof [Viz64]. Couched in our language, this is closely related to designing a color extension subroutine with small *cost*. In particular, for a $(\Delta + 1)$-color extension subroutine, the current state-of-the-art bounds are by [Chr23] and [Ber22], who respectively achieve $O(\Delta^7 \log n)$ and $O(\Delta^6 \log^2 n)$ costs using multi-step Vizing chains. By designing an optimized multi-step Vizing chain construction, we obtain an improved algorithm for this task, proving the following theorem.

Theorem 5.1.2. *Given a graph $G = (V, E)$ and a partial $(\Delta + 1)$-edge coloring χ of G with an uncolored edge $e \in E$, we can extend χ to the edge e in $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$ expected time.*

The starting point of our approach for proving Theorem 5.1.2 is the multi-step Vizing chain construction of [DHZ19]. Specifically, [DHZ19] designed a $((1 + \epsilon)\Delta)$-color extension subroutine

with an expected runtime of $\tilde{O}(1/\epsilon^2)$, as long as $\Delta = \tilde{\Omega}(1)$ and $\epsilon = \tilde{\Omega}(1/\sqrt{\Delta})$. A priori, it seems that such a bound will not be suitable for our purpose, because we want to set $\epsilon = 1/\Delta$. We now outline, at a very high level, the algorithm of [DHZ19] and how we build on top of it.

In the ensuing paragraphs, we assume that the reader is already familiar with the proof of Vizing's theorem and concepts like "alternating paths", "fans" and "Vizing chains" (see Section 2.1).

The [DHZ19] Algorithm. We are given an input graph $G = (V, E)$, and a partial $(1+\epsilon)\Delta$-edge coloring χ in G with one uncolored edge $e = (u, v)$. The algorithm of [DHZ19] proceeds in $R = \Theta(\log n)$ *rounds*. For each round $i \in \{1, \ldots, R\}$, it samples a subset of colors $\mathcal{C}_i \subseteq [\Delta+1] \setminus \left(\bigcup_{j<i} \mathcal{C}_j\right)$ of size $|\mathcal{C}_i| = \Theta(\log n/\epsilon)$ independently and u.a.r. We refer to $\mathcal{C}_i$ as the *palette* for round i. For technical reasons, [DHZ19] want these palettes across different rounds to be mutually disjoint. In addition, for each $i \in [R]$, they want that under any fixed partial coloring χ' of G, every vertex v has at least one *missing color* in $\mathcal{C}_i$.[1] To ensure these two properties, it is easy to verify that we must have $|R| \cdot |\mathcal{C}_i| = \Theta(\log^2 n/\epsilon) < \epsilon\Delta$, the RHS being the *slack*, in terms of the number of extra available colors. This necessitates the requirement that $\epsilon^2 = \tilde{\Omega}(1/\Delta)$, and hence $\epsilon = \tilde{\Omega}(1/\sqrt{\Delta})$.

Let $u_1 := u$, $v_1 := v$, $e_1 := (u_1, v_1)$ and $\chi_1 := \chi$. At the start of a given round $i \in [R]$, [DHZ19] have a partial coloring χ_i and an uncolored edge $e_i = (u_i, v_i)$ w.r.t. χ_i. Using only the colors from the palette $\mathcal{C}_i$, they identify a Vizing chain starting from an endpoint (say) u_i of e_i. If the alternating path corresponding to this Vizing chain has length less than $L := \tilde{\Theta}(1/\epsilon^2)$, then they *apply* the Vizing chain to extend the partial coloring to e_i, and their algorithm terminates. Otherwise, they pick some $\alpha_i \in [L]$ independently and u.a.r., and *shift* the position of the uncolored edge *from* e_i *to* the α_i^{th} edge (say) $e_{i+1} = (u_{i+1}, v_{i+1})$ on the concerned alternating path (say) P_i. This is done by appropriately shifting the colors on the Vizing fan and the first α_i edges of P_i, which leads to a new partial coloring χ_{i+1}. The algorithm then proceeds to round $(i+1)$.

The choice of the value of L is dictated by the fact that for technical reasons, [DHZ19] have to ensure that $L = \Omega(|\mathcal{C}_i|^2)$. The main technical contribution of [DHZ19] is to show that this algorithm terminates within R rounds, w.h.p.[2]

Our Approach. At a very high-level, we diverge from [DHZ19] in two major aspects. First, since we need to set $\epsilon = 1/\Delta$, we cannot afford to have these separate palettes $\{\mathcal{C}_i\}_i$ across different rounds, and so we get rid of them altogether. Morally, we observe that all we need is the following *key property*: The pair of colors on the concerned alternating path in each round $i \in [R]$ is disjoint from the ones used in previous rounds. As long as this key property holds, the [DHZ19] analysis continues to work just fine. (This assertion has a big caveat associated with it, namely, the complications that arise from the Vizing fans; see Section 5.2.5 for a discussion on those complications. But we ignore the fans for now.)

The second point of divergence from [DHZ19] arises because the key property might not hold

[1] We say that a color is missing at v if it is *not* assigned to any of the edges incident on v. Since v has degree at most Δ, it has at least $\epsilon\Delta$ missing colors.

[2] The in-expectation guarantee follows because if the algorithm fails, then it can always revert back to the trivial implementation of Vizing's proof to extend the partial coloring to one edge in $O(n)$ time.

after a certain number of rounds. To address this issue, our main insight is to increase the value of the parameter L to be $:= \tilde{\Theta}(1/\epsilon^2 + \sqrt{\Delta n}) = \tilde{\Theta}(\Delta^2 + \sqrt{\Delta n})$. With this new increased value of L, we derive the following crucial implication (see Lemma 5.2.4). Let $i \in [R]$ be the first round such that: most of the L choices for the value of α_i lead to a scenario where the subsequent round $i+1$ will not satisfy the key property. Then with sufficiently large probability, the algorithm will terminate in the next round (i.e., in round $i+1$). This leads us to a *win-win* framework. Either round i is such that with good probability we can continue to apply the analysis of [DHZ19] in the subsequent round, or with good probability our algorithm actually terminates in the subsequent round. Section 5.2 contains a much more detailed exposition of this analysis.

Finally, we need to overcome further significant obstacles to deal with the Vizing fans. The complete proof of Theorem 5.1.2, which handles these obstacles, is deferred to Chapter 9.[3]

5.2 Overview of Our Multi-Step Vizing Chain Algorithm

We define the parameters:

$$\ell := 100 \log n, \quad L := 10^3 \ell^2 (\Delta^2 + \sqrt{\Delta n}) \tag{5.1}$$

To convey the main conceptual ideas behind our analysis, in this section we will explain the proof of Theorem 5.1.2 under Assumption 5.2.1, stated below. This assumption clearly holds, for example, on bipartite graphs. For those readers familiar with the proof of Vizing's theorem, this assumption will allow us to ignore the Vizing fans altogether, and we will be able to focus only on the alternating paths, which are more intuitive to reason about.

Assumption 5.2.1. *The graph $G = (V, E)$ does not contain any odd cycle of length $\leq 2L + 3$.*

In Section 5.2.1, we introduce some key notations and terminologies regarding alternating paths. Next, we describe our randomized algorithm in Section 5.2.2. In Section 5.2.3, we introduce a recursion tree (which we refer to as the "meta-tree") which encodes all possible execution-paths of our algorithm, and present a few basic properties of this meta-tree. We show that our algorithm essentially performs a random walk on this meta-tree, and state Lemma 5.2.2 which guarantees that this random walk terminates quickly. We prove Lemma 5.2.2, which implies Theorem 5.1.2, in Section 5.2.4. Finally, we conclude our discussion in Section 5.2.5 by pointing out the remaining significant technical challenges that we need to overcome, if we are to get rid of Assumption 5.2.1.

5.2.1 Preliminaries

We now provide the notation that we use to describe our algorithm and also recap some of the relevant basic notation from Section 3.1.

[3] The proof in Chapter 9 is self-contained and uses slightly different notation than the proof sketch in Section 5.2.

Alternating Paths and Types. Consider a partial $(\Delta+1)$-edge coloring $\chi : E \to [\Delta+1] \cup \{\perp\}$ in the input graph $G = (V, E)$. Consider a path $P = ((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k))$ in G, where $(v_{i-1}, v_i) \in E$ is the i^{th} edge on the path, for all $i \in [k]$. We say that P is an **alternating path** (w.r.t. χ) iff there exist two distinct colors $c, c' \in [\Delta+1]$ that satisfy the following conditions.

1. The colors on the consecutive edges of P alternate between c and c'. Specifically, this means that $\chi(v_{i-1}, v_i) \in \{c, c'\}$ for all $i \in [k]$, and $\chi(v_{i-1}, v_i) \neq \chi(v_i, v_{i+1})$ for all $i \in [k-1]$.
2. The path P is *maximal*. Thus, for each vertex $u \in \{v_0, v_k\}$, either $c \in \mathsf{miss}_\chi(u)$ or $c' \in \mathsf{miss}_\chi(u)$, where $\mathsf{miss}_\chi(u) \subseteq [\Delta+1]$ denotes the set of *missing colors* at u under χ (i.e., these are the colors that are *not* assigned to any edge in G incident on u).

Let $\tau = \{c, c'\}$. We refer to τ as being the **type** of the alternating path P. We let $\texttt{length}(P) = k$ denote the length of the path P. We also say that the path P **starts** at v_0 and **ends** at v_k. For $i \in [k]$, we let $P_{\leq i} = ((v_0, v_1), (v_1, v_2), \ldots, (v_{i-1}, v_i))$ denote the **length-i prefix** of P. For $i > k$, we let $P_{\leq i} := P$. Finally, note that an alternating path always comes with an associated **orientation**. In particular, there is another alternating path with the same set of edges as P, but in reverse order.

Our algorithm in Section 5.2.2 will use two basic subroutines, as described below.

The Subroutine $\mathsf{Apply}(\chi, e, P)$. Here, the input is a partial $(\Delta+1)$-edge coloring χ of G, an uncolored edge $e = (u, v)$ and an alternating path P starting from u that is of type $\tau = \{\alpha_u, \alpha_v\}$, where $\alpha_u \in \mathsf{miss}_\chi(u) \setminus \mathsf{miss}_\chi(v)$ and $\alpha_v \in \mathsf{miss}_\chi(v) \setminus \mathsf{miss}_\chi(u)$. (If either $\alpha_u \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$ or $\alpha_v \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$, our algorithm will not make the call $\mathsf{Apply}(\chi, e, P)$.) Crucially, it is guaranteed that $\texttt{length}(P) \leq 2L+2$. W.l.o.g., suppose that $P = ((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k))$, where $k = \texttt{length}(P)$ and $v_0 = u$, and let $c_k = \chi(v_{k-1}, v_k) \in \{\alpha_u, \alpha_v\}$. It is easy to verify that under Assumption 5.2.1, we have $v_k \neq v$, for otherwise the path P along with the edge (u, v) would create an odd cycle of length $\leq 2L+3$. The subroutine updates the coloring χ as follows.

- $\chi(v_{i-1}, v_i) \leftarrow \chi(v_i, v_{i+1})$ for all $i \in [k-1]$.
- $\chi(v_{k-1}, v_k) \leftarrow c$, where c is the unique color in $\{\alpha_u, \alpha_v\} \setminus \{c_k\}$.
- $\chi(u, v) \leftarrow \alpha_v$.

At the end of the above operations, the partial coloring χ gets extended to the edge e. The subroutine returns the updated partial coloring. In summary, the subroutine *applies* the alternating path P to extend the partial coloring to e.

The Subroutine $\mathsf{Shift}(\chi, e, P_{\leq i})$. Here, the input is a partial $(\Delta+1)$-edge coloring χ of G, an uncolored edge $e = (u, v)$ and a length-i prefix $P_{\leq i}$ of an alternating path P starting from u that is of type $\tau = \{\alpha_u, \alpha_v\}$, where $\alpha_u \in \mathsf{miss}_\chi(u) \setminus \mathsf{miss}_\chi(v)$ and $\alpha_v \in \mathsf{miss}_\chi(v) \setminus \mathsf{miss}_\chi(u)$. (If either $\alpha_u \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$ or $\alpha_v \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$, our algorithm will not make the call $\mathsf{Shift}(\chi, e, P_{\leq i})$.) Crucially, we are also guaranteed that $i \leq 2L+2$. W.l.o.g., let $P_{\leq i} = ((v_0, v_1), (v_1, v_2), \ldots, (v_{i-1}, v_i))$, where $v_0 = u$. Due to Assumption 5.2.1, it is again easy to verify that $v_i \neq v$. The subroutine updates χ as follows.

- $\chi(v_{j-1}, v_j) \leftarrow \chi(v_j, v_{j+1})$ for all $j \in [i-1]$.
- $\chi(v_{i-1}, v_i) \leftarrow \perp$.
- $\chi(u, v) \leftarrow \alpha_v$.

At the end of the above operations, in the partial coloring χ the position of the uncolored edge gets *shifted* from e to (v_{i-1}, v_i). The subroutine returns the updated coloring.

5.2.2 Our Algorithm

We have a graph $G = (V, E)$, and a partial $(\Delta+1)$-edge coloring χ of G with one uncolored edge $e = (u, v)$. We need to extend χ to e. We do this by identifying two colors $c_u \in [\Delta+1] \setminus \mathsf{miss}_\chi(u)$ and $c_v \in [\Delta+1] \setminus \mathsf{miss}_\chi(v)$, and then calling Algorithm 5 with input $(G, \chi, e = (u, v), \{c_u, c_v\})$.

While parsing the pseudocode of Algorithm 5, the reader should think of c_u and c_v as **blocking colors**. The algorithm considers an alternating path P starting from u, such that the type of this alternating path, given by $\{\alpha_u, \alpha_v\}$, is disjoint from $\{c_u, c_v\}$. Note that it is always possible to find such a type, for the following reason. Since u has degree at most Δ in G and the edge $e = (u, v)$ is currently uncolored, we have $|\mathsf{miss}_\chi(u)| \geq (\Delta+1) - (\Delta-1) = 2$. As $c_u \notin \mathsf{miss}_\chi(u)$, it must be the case that $\mathsf{miss}_\chi(u) \setminus \{c_u, c_v\} \neq \emptyset$, and so there exists an appropriate color α_u that we can pick from $\mathsf{miss}_\chi(u) \setminus \{c_u, c_v\}$. A similar argument holds for α_v. Also, by induction, it is easy to verify that all future recursive calls to the algorithm will continue to satisfy the same property with regard to the blocking colors. Specifically, if the algorithm is called with input $(G, \chi, (u', v'), \{c_1, c_2\})$, then $\mathsf{miss}_\chi(w) \cap \{c_1, c_2\} \neq \emptyset$ for each $w \in \{u', v'\}$. The reason for having these blocking colors will become apparent later on (see Observation 5.2.3 and the proof of Lemma 5.2.4).

Algorithm 5: $\mathsf{ExtendColoring}(G, \chi, e = (u, v), \{c_u, c_v\})$

```
Find two colors α_u ∈ miss_χ(u) \ {c_u, c_v} and α_v ∈ miss_χ(v) \ {c_u, c_v}
if ∃c ∈ {α_u, α_v} such that c ∈ miss_χ(u) ∩ miss_χ(v) then
    χ(u, v) ← c
    return χ
Let P := (e_1, ..., e_k) be the {α_u, α_v}-alternating path in G (w.r.t. χ) starting from u
if k ≤ 2L + 2 then
    χ ← Apply(χ, e, P)
    return χ
else
    Sample i ∈ {1, ..., L} independently and u.a.r.
    Let P_{≤i} = (e_1, ..., e_i) denote the prefix of P consisting of its first i edges
    χ ← Shift(χ, e, P_{≤i})
    ExtendColoring(G, χ, e_i, {α_u, α_v})
```

The remainder of Algorithm 5 is very natural and intuitive. If there exists some color $c \in \{\alpha_u, \alpha_v\}$ that is missing *both* at u and v, then the algorithm simply assigns the color c to e, and terminates. Otherwise, if the length of the alternating path P is at most $2L+2$, then it extends

the partial coloring to e by applying the alternating path, and terminates. Finally, if $\texttt{length}(P) > 2L+2$, then it picks some $i \in [L]$ u.a.r., shifts the position of the uncolored edge from e to the i^{th} edge on P, and makes a recursive call to itself.

5.2.3 The Meta-Tree

We now define a (possibly infinite) tree $\mathcal{T}$ that captures all possible execution paths taken by our recursive algorithm, on a given input. To clearly distinguish it from the input graph G, we refer to $\mathcal{T}$ as a **meta-tree** and its vertices as **meta-nodes**. We next introduce some key notations.

The meta-tree $\mathcal{T}$ is rooted at a meta-node r. We let $V(\mathcal{T})$ denote the set of all meta-nodes in $\mathcal{T}$. Every meta-node $x \in V(\mathcal{T})$ corresponds to a recursive **call** of Algorithm 5, and the root-to-leaf path from r to x in $\mathcal{T}$ corresponds to the execution of our recursive algorithm leading to this call.[4] For each such meta-node x, we denote the state of an entity Φ at the start of the corresponding call of Algorithm 5 by $\Phi^{(x)}$. For example, we use the symbols $\chi^{(x)}$, $P^{(x)}$ and $\tau^{(x)}$ respectively to denote the following entities at the start of the concerned call of Algorithm 5: (i) the current partial coloring χ, (ii) the alternating path P (see Line 5 of Algorithm 5) and (iii) the type $\{\alpha_u, \alpha_v\}$ of the path P (see Line 5 of Algorithm 5).

If a meta-node x is *not* a leaf in $\mathcal{T}$, then it has exactly L children, one for each choice of $i \in [L]$ in Line 10 of Algorithm 5. In contrast, a meta-node x is a leaf in $\mathcal{T}$ iff the corresponding call of Algorithm 5 terminates at either Line 4 or Line 8. We will refer to the leaves in $\mathcal{T}$ as **terminals**.

Random Walks in $\mathcal{T}$. An execution of our algorithm defines a **random walk** on $\mathcal{T}$ that starts at the root and, at each step, independently and uniformly samples a random child of the current meta-node. The random walk ends when, and if, it reaches a terminal meta-node: At that point the algorithm succeeds in extending the initial partial coloring χ to the uncolored edge given to it as part of the input. Using standard data structures, it is easy to ensure that the algorithm spends $\tilde{O}(L)$ time at each meta-node it visits during this random walk, because the colors of at most $O(L)$ edges get changed during any given call of Algorithm 5. Since $L = \tilde{O}(\Delta^2 + \sqrt{\Delta n})$, Theorem 5.1.2 follows from Lemma 5.2.2 and standard tools for boosting the success probability of a randomized algorithm. We derive some basic properties of the meta-tree in Section 5.2.3. Subsequently, we use these properties to prove Lemma 5.2.2 in Section 5.2.4.

Lemma 5.2.2. *With probability $\Omega(1/\log n)$, the random walk on $\mathcal{T}$ executed by our algorithm ends at a terminal vertex that lies within depth $O(\log n)$ from the root r.*

Basic Properties of the Meta-Tree

Recall that $\tau^{(x)}$ denotes the type of the alternating path $P^{(x)}$ at a meta-node x. Consider a non-terminal meta-node x at **depth** $= k$ (say), and suppose that $(x_0, x_1, \ldots, x_k)$ is the unique path in $\mathcal{T}$ from the root to x (i.e., $r = x_0$ and $x = x_k$). We say that the meta-node x is **dirty** iff

[4] While giving the full proof in Chapter 9, we formulate our algorithm iteratively instead of recursively and refer to **iterations** instead of **calls**.

$\tau^{(x)} \cap \left(\tau^{(x_0)} \cup \cdots \cup \tau^{(x_{k-1})}\right) \neq \emptyset$, and **clean** otherwise. We also say that x is **contaminated** iff at least $L/(10\ell)$ of its children are dirty. Note that a contaminated meta-node itself might be clean.

Remark. We emphasize that according to our definitions *only* the non-leaf meta-nodes are classified as being either clean or dirty. Thus, the set of meta-nodes is partitioned into three subsets: terminal, dirty and clean. Furthermore, a subset of non-terminal meta-nodes are contaminated. This implies that some of the contaminated meta-nodes are clean, the rest being dirty.

We next derive a few key properties that will be useful in proving Lemma 5.2.2 later on.

Observation 5.2.3. *Consider any non-terminal meta-node $x \in V(\mathcal{T})$, and let y be any child of x. If y is* not *a terminal, then we must have $\tau^{(x)} \cap \tau^{(y)} = \emptyset$.*

Proof. Follows from Line 1 and Line 13 of Algorithm 5. □

Lemma 5.2.4. *Consider any contaminated meta-node $x \in V(\mathcal{T})$ at a depth $\leq \ell$ in the meta-tree $\mathcal{T}$. Then at least $L/(40\ell)$ many children of x are terminal meta-nodes.*

Proof. Let $(x_0, x_1, \ldots, x_k)$ denote the unique path from the root to x in $\mathcal{T}$, with $r = x_0$ and $x = x_k$. Thus, at most $2(k+1) \leq 2(\ell+1) \leq 4\ell$ distinct colors appear in the set $\tau^{(x_0)} \cup \tau^{(x_1)} \cup \cdots \cup \tau^{(x_k)} = \mathcal{C}^\star$ (say); because $|\tau^{(x_i)}| = 2$ for all $i \in [k]$. Let $K^\star := \{\tau \in \binom{[\Delta+1]}{2} : \tau \cap \mathcal{C}^\star \neq \emptyset\}$ denote the collection of types with at least one color from $\mathcal{C}^\star$. Note that $|K^\star| \leq |\mathcal{C}^\star|(\Delta+1) \leq 4\ell(\Delta+1) \leq 8\ell\Delta$. Let D denote the set of dirty children of x. By definition, for each meta-node $y \in D$, we have $\tau^{(y)} \in K^\star$. Furthermore, since x is contaminated, it follows that $|D| \geq L/(10\ell)$.

Say that a meta-node is **trivial** iff the corresponding call of Algorithm 5 ends at Line 4. Thus, every trivial meta-node is terminal, but not vice versa. Let $D_t \subseteq D$ denote the set of trivial meta-nodes that belong to D. If at least half of the meta-nodes in D are trivial, then $|D_t| \geq |D|/2 \geq L/(20\ell)$, and the lemma follows. Thus, for the rest of the proof we assume that:

$$|D \setminus D_t| \geq |D|/2 \geq L/(20\ell). \tag{5.2}$$

Since $\tau^{(y)} \cap \tau^{(x)} = \emptyset$ for all $y \in D \setminus D_t$ (see Observation 5.2.3), all the alternating paths in the collection $\mathcal{P}' := \{P^{(y)}\}_{y \in D \setminus D_t}$ exist *simultaneously* in the graph G w.r.t. the partial $(\Delta+1)$-edge coloring $\chi^{(x)}$. To be more precise, this means that for every meta-node $y \in D \setminus D_t$ and every edge $e \in P^{(y)}$, we have $\chi^{(y)}(e) = \chi^{(x)}(e)$. Furthermore, we have already inferred that each path $P^{(y)} \in \mathcal{P}'$ is of a type $\tau^{(y)} \in K^\star$. Next, note that the total length of all possible alternating paths of a given type (w.r.t. a specific partial coloring) is at most $2n$. This holds because such paths are vertex-disjoint, except the same path being possibly counted twice from two opposite directions. Thus, we have $\sum_{P \in \mathcal{P}'} \texttt{length}(P) \leq 2n|K^\star| \leq 16\ell\Delta n$. Hence, the average length of an alternating path $P^{(y)} \in \mathcal{P}'$ is at most $16\ell\Delta n/|\mathcal{P}'| = 16\ell\Delta n/|D \setminus D_t| \leq 320\ell^2\Delta n/L \leq L$, where the second-last inequality follows from (5.2) and the last inequality[5] follows from (5.1). This implies that at least half of the alternating paths in $\mathcal{P}'$ have length at most $2L$. So, at least half of the meta-nodes

[5]This is the only place in our analysis where we require L to be $\tilde{\Omega}(\sqrt{\Delta n})$.

$y \in D \setminus D_t$ have $\texttt{length}\left(P^{(y)}\right) \leq 2L$; and such meta-nodes are terminals. We therefore conclude that at least $|D \setminus D_t|/2 \geq L/(40\ell)$ children of x are terminal meta-nodes. □

We say that a meta-node $x \in V(\mathcal{T})$ is **congenitally clean** iff every ancestor of x, along with x itself, is clean. Next, consider any meta-node $x \in V(\mathcal{T})$ at depth (say) k in $\mathcal{T}$. Let $(x_0, x_1, \ldots, x_k)$ denote the unique path from the root to x in $\mathcal{T}$, with $r = x_0$ and $x = x_k$. Then we refer to the ordered tuple of types $\left(\tau^{(x_0)}, \tau^{(x_1)}, \ldots, \tau^{(x_k)}\right)$ as the **transcript** of x. We now upper bound the number of congenitally clean vertices with the same transcript.

Lemma 5.2.5. *Let $\tau_0, \ldots, \tau_i$ be a sequence of types. Then there can be at most n congenitally clean meta-nodes with transcript $= (\tau_0, \ldots, \tau_i)$.*

Proof. For $j \in [0, i]$, let $\Gamma_j \subseteq V(\mathcal{T})$ denote the set of congenitally clean meta-nodes with transcript $= (\tau_0, \ldots, \tau_j)$. Note that every meta-node in Γ_j is at depth $= j$ in $\mathcal{T}$.

Claim 5.2.6. *For all $j \in [0, i]$, the collection $\left\{P^{(x)}_{\leq L}\right\}_{x \in \Gamma_j}$ of length-L prefixes are vertex-disjoint.*

Setting $j = i$ in Claim 5.2.6, it follows that the size of the set Γ_i is at most the maximum possible number of vertex-disjoint paths in the input graph G, which in turn, is at most n. This implies the lemma. Accordingly, from now on we focus on proving Claim 5.2.6.

We will prove Claim 5.2.6 via induction on j. Since $\Gamma_0 = \{r\}$, the claim trivially holds if $j = 0$. By induction hypothesis, we now assume that there exists an index $j^\star \in [0, i-1]$ such that the claim holds for all $j \leq j^\star$. Under this assumption, we will show that the claim holds for $j = j^\star + 1$.

If there is a color that appears more than once across the types $\tau_0, \ldots, \tau_{j^\star+1}$, then a meta-node with the transcript $(\tau_0, \ldots, \tau_{j^\star+1})$ is not congenitally clean, and so $\Gamma_j = \emptyset$ for all $j \in [j^\star + 1, i]$. Henceforth, we assume that the types $\tau_0, \ldots, \tau_{j^\star+1}$ are mutually disjoint.

Consider any two distinct meta-nodes $x, y \in \Gamma_{j^\star+1}$. Let $u^{(x)}$ and $u^{(y)}$ respectively denote the starting points of the alternating paths $P^{(x)}$ and $P^{(y)}$. Our induction hypothesis implies that $u^{(x)} \neq u^{(y)}$. Since the types $\tau_0, \ldots, \tau_{j^\star+1}$ are mutually disjoint, both the alternating paths $P^{(x)}$ and $P^{(y)}$ exist *simultaneously* in G w.r.t. the initial partial coloring $\chi^{(r)}$. In other words, either they are two completely disjoint type-$\tau_{j^\star+1}$ alternating paths in G w.r.t. $\chi^{(r)}$, or essentially the same path (with the same set of edges) but with a different orientation. In the first case, their length-L prefixes $P^{(x)}_{\leq L}$ and $P^{(y)}_{\leq L}$ are clearly vertex-disjoint. In the second case, we note that x and y are *not* terminal meta-nodes (because every meta-node in $\Gamma_{j^\star+1}$ is congenitally clean, by definition); hence both $P^{(x)}$ and $P^{(y)}$ have length $\geq 2L + 2$, and so $P^{(x)}_{\leq L}$ and $P^{(y)}_{\leq L}$ are also vertex-disjoint. This concludes the proof of the claim. □

5.2.4 Analyzing the Random Walk on the Meta-Tree: Proof of Lemma 5.2.2

Our analysis will crucially rely on the behavior of the random walk within a certain **critical subtree** $\mathcal{T}^\star$ of $\mathcal{T}$. We define this critical subtree below, and then summarize a few of its key properties.

The "critical subtree" $\mathcal{T}^\star$ is obtained by starting with $\mathcal{T}$, and then deleting every meta-node $x \in V(\mathcal{T})$ that satisfies at least one of the following conditions: (i) x has a dirty ancestor in $\mathcal{T}$, (ii) x has a contaminated ancestor in $\mathcal{T}$, and (iii) x is at a depth strictly greater than ℓ in $\mathcal{T}$. We let $V(\mathcal{T}^\star) \subseteq V(\mathcal{T})$ denote the set of meta-nodes in the critical subtree.

Note that if the root r is itself a terminal, then Lemma 5.2.2 trivially holds because the random walk ends at r. Thus, for the rest of the proof we assume that **the root r is *not* a terminal**.

Observation 5.2.7. *$\mathcal{T}^\star$ is a connected subtree of $\mathcal{T}$, rooted at r, and r is* not *a leaf in $\mathcal{T}^\star$. Also, for every $x \in V(\mathcal{T}) \setminus V(\mathcal{T}^\star)$, the unique path in $\mathcal{T}$ from r to x passes through some leaf in $\mathcal{T}^\star$.*

Proof. Since the r is at depth 0 and does not have any ancestor, it belongs to $\mathcal{T}^\star$. Furthermore, we have assumed that r is *not* a terminal. This implies that, by definition, r is a congenitally clean meta-node. So all the children of r are part of $\mathcal{T}^\star$, and hence r is *not* a leaf in $\mathcal{T}^\star$.

Next, note that if a meta-node x is in $\mathcal{T}^\star$, then each of its siblings and each of its ancestors is also in $\mathcal{T}^\star$. This implies the observation. □

Observation 5.2.8. *Every non-leaf meta-node in $\mathcal{T}^\star$ is congenitally clean and not contaminated.*

Proof. Consider any meta-node $x \in V(\mathcal{T}^\star)$. If x is terminal, then it is a leaf in $\mathcal{T}$ itself, and hence also a leaf in $\mathcal{T}^\star$. In contrast, if x is either contaminated or dirty, then it cannot have any descendant in $\mathcal{T}^\star$. Thus, for x to be a non-leaf meta-node in $\mathcal{T}^\star$, it must be clean and not contaminated. Since we can infer the same for every ancestor of x, the observation follows. □

The above observations help us gain an intuitive understanding of the critical subtree $\mathcal{T}^\star$. Define the **core** (resp. **boundary**) of $\mathcal{T}^\star$ to be the set of its non-leaf (resp. leaf) meta-nodes. Every meta-node within the core is congenitally clean and not contaminated (see Observation 5.2.8). The random walk on $\mathcal{T}$ undertaken by our algorithm starts at the root r, which is part of the core of $\mathcal{T}^\star$ (see Observation 5.2.7). For a certain number of steps the random walk stays within the core. After that, at some point in time the random walk exits the core by reaching a meta-node (say) x at the boundary of $\mathcal{T}^\star$ (see Observation 5.2.7). If x is terminal, then the random walk ends at x. Otherwise, it ventures out of $V(\mathcal{T}^\star)$ in the subsequent step, and never comes back to $V(\mathcal{T}^\star)$ in future.

The above discussion also implies that for every meta-node $x \in V(\mathcal{T}^\star)$, its depth in $\mathcal{T}^\star$ is the same as its depth in $\mathcal{T}$. Accordingly, from this point onward we will use the phrase "the depth of a meta-node" without explicitly referring to the underlying meta-tree.

Observation 5.2.9. *Let $\mathcal{Z}^\star$ denote the set of leaves in $\mathcal{T}^\star$. Then $\mathcal{Z}^\star$ is partitioned into four subsets:*

- $\mathcal{Z}^\star_{\texttt{t}} := \{x \in \mathcal{Z}^\star : x \text{ is terminal}\}$.
- $\mathcal{Z}^\star_{\texttt{cc-cont}} := \{x \in \mathcal{Z}^\star : x \text{ is congenitally clean and contaminated}\}$.

- $\mathcal{Z}^\star_{\texttt{cc-not-cont}} := \{x \in \mathcal{Z}^\star : x \text{ is congenitally clean and } \textit{not} \text{ contaminated}\}$.
- $\mathcal{Z}^\star_{\texttt{d}} := \{x \in \mathcal{Z}^\star : x \text{ is dirty}\}$.

Proof. This holds because the set of meta-nodes is partitioned into three substes: terminal, clean and dirty. Finally, every clean meta-node in $\mathcal{T}^\star$ is congenitally clean, because it cannot have any dirty ancestor. □

Corollary 5.2.10. *Let $\mathcal{E}^\star_{d}, \mathcal{E}^\star_{t}, \mathcal{E}^\star_{\texttt{cc-cont}}$ and $\mathcal{E}^\star_{\texttt{cc-not-cont}}$ respectively denote the events that the random walk undertaken by our algorithm reaches a meta-node in $\mathcal{Z}^\star_{d}, \mathcal{Z}^\star_{t}, \mathcal{Z}^\star_{\texttt{cc-cont}}$ and $\mathcal{Z}^\star_{\texttt{cc-not-cont}}$. These four events are mutually exclusive and exhaustive, and hence:*

$$\Pr[\mathcal{E}^\star_{d}] + \Pr[\mathcal{E}^\star_{t}] + \Pr[\mathcal{E}^\star_{\texttt{cc-cont}}] + \Pr[\mathcal{E}^\star_{\texttt{cc-not-cont}}] = 1.$$

Proof. Follows from Observation 5.2.7 and Observation 5.2.9. □

Observation 5.2.11. *Every meta-node in $\mathcal{Z}^\star_{\texttt{cc-not-cont}}$ is at depth ℓ.*

Proof. Consider any meta-node $x \in \mathcal{Z}^\star_{\texttt{cc-not-cont}}$. By definition, the depth of x is no more than ℓ. Suppose that x is at a depth (say) $k < \ell$. Since x is clean, it has L children in the meta-tree $\mathcal{T}$.

Let y be any child of x in $\mathcal{T}$. Since x is congenitally clean, y does not have any dirty ancestor. Also, since x is *not contaminated*, y cannot have any contaminated ancestor; for otherwise x itself would have had the same contaminated ancestor and so x would not be part of $\mathcal{T}^\star$. Finally, the meta-node y is at depth $= k+1 \leq \ell$. We therefore conclude that y is part of $\mathcal{T}^\star$. But this contradicts our assumption that x is a leaf in $\mathcal{T}^\star$. So, the meta-node x must be at depth $k = \ell$. □

Armed with these basic observations about the critical subtree, we are now ready to prove Lemma 5.2.2. Our strategy will be to show that the event $\mathcal{E}^\star_{\texttt{cc-cont}} \cup \mathcal{E}^\star_{t}$ occurs with constant probability, and conditioned on this event, the random walk ends within $O(\log n)$ steps with probability $\Omega(1/\log n)$.

Claim 5.2.12. *We have* $\Pr[\mathcal{E}^\star_{\texttt{cc-not-cont}}] \leq 1/10$.

Proof. By Observation 5.2.11, every meta-node $x \in \mathcal{Z}^\star_{\texttt{cc-not-cont}}$ is at depth ℓ. As there are $\binom{\Delta+1}{2} \leq \Delta^2$ possible types, the meta-nodes at depth ℓ have at most $(\Delta^2)^{\ell+1} = \Delta^{2(\ell+1)}$ possible transcripts. For each transcript, there are at most n meta-nodes in $\mathcal{Z}^\star_{\texttt{cc-not-cont}}$ (see Lemma 5.2.5). Thus, we get:

$$|\mathcal{Z}^\star_{\texttt{cc-not-cont}}| \leq \Delta^{2(\ell+1)} n.$$

Next, consider any meta-node $x \in \mathcal{Z}^\star_{\texttt{cc-not-cont}}$, and let $(x_0, x_1, \ldots, x_\ell)$ be the unique path from the root r to x in $\mathcal{T}$ (and in $\mathcal{T}^\star_\ell$), with $r = x_0$ and $x = x_\ell$. By our construction of the meta-tree $\mathcal{T}$, every internal meta-node on this path has exactly L children in $\mathcal{T}$. Thus, the random walk taken

by our algorithm traverses this path (and ends up at x) with probability $1/L^\ell$. Summing these probabilities over all $x \in \mathcal{Z}^\star_{\texttt{cc-not-cont}}$, we get:

$$\Pr\left[\mathcal{E}^\star_{\texttt{cc-not-cont}}\right] \leq \left|\mathcal{Z}^\star_{\texttt{cc-not-cont}}\right| \cdot (1/L^\ell) \leq \Delta^{2(\ell+1)} n/L^\ell < 1/10,$$

where the last inequality follows from (5.1). □

Claim 5.2.13. *We have* $\Pr\left[\mathcal{E}^\star_{\texttt{d}}\right] \leq 1/5$.

Proof. Consider any $i \in [0, \ell-1]$. Let $\mathcal{E}_i$ denote the event that after i recursive calls of Algorithm 5, the concerned random walk is at a meta-node (say) x_i that belongs to the core of $\mathcal{T}^\star$, and as a corollary, the event $\mathcal{E}^\star_{\texttt{d}}$ has not yet taken place. By Observation 5.2.7, we have:

$$\Pr[\mathcal{E}_0] = 1. \tag{5.3}$$

We will next prove the following inequality.

$$\Pr\left[\,\overline{\mathcal{E}^\star_d} \cup \mathcal{E}_{i+1} \,\middle|\, \mathcal{E}_i\,\right] \geq 1 - \frac{1}{10\ell} \text{ for all } i \in [0, \ell-1]. \tag{5.4}$$

Towards this end, fix any $i \in [0, \ell-1]$, and condition on the event $\mathcal{E}_i$. Since x_i is part of the core of $\mathcal{T}^\star$, it is congenitally clean and not contaminated (see Observation 5.2.8). Thus, the meta-node x_i has L children in $\mathcal{T}$ and at most $L/(10\ell)$ of these children are dirty. Accordingly, with probability at least $1 - 1/(10\ell)$, in the very next step the random walk moves on to a non-dirty child (say) y of x. Such a non-dirty child y is either: (1) terminal, or (2) congenitally clean and contaminated, or (3) congenitally clean and not contaminated. In the former two cases, we are guaranteed that the event $\mathcal{E}^\star_{\texttt{d}}$ cannot occur, because the event $\mathcal{E}^\star_{\texttt{t}} \cup \mathcal{E}^\star_{\texttt{cc-cont}}$ has already taken place and this is mutually exclusive with the event $\mathcal{E}^\star_{\texttt{d}}$ (see Corollary 5.2.10). We next consider the third case. Here, if $i < \ell-1$, then meta-node y is still within the core of $\mathcal{T}^\star$, and hence the event $\mathcal{E}_{i+1}$ has occurred. Otherwise, if $i = \ell - 1$, then the event $\mathcal{E}^\star_{\texttt{cc-not-cont}}$ has occurred which again is mutually exclusive with the event $\mathcal{E}^\star_d$ (see Corollary 5.2.10). This concludes the proof of inequality (5.4).

From (5.3) and (5.4), we infer that $\Pr\left[\,\overline{\mathcal{E}^\star_d}\,\right] \geq (1 - 1/(10\ell))^\ell \geq 4/5$. Thus, we get $\Pr\left[\,\mathcal{E}^\star_d\,\right] = 1 - \Pr\left[\,\overline{\mathcal{E}^\star_d}\,\right] \leq 1/5$. □

Corollary 5.2.14. *We have* $\Pr\left[\mathcal{E}^\star_t\right] + \Pr\left[\mathcal{E}^\star_{\texttt{cc-cont}}\right] \geq 7/10$.

Proof. Follows from Corollary 5.2.10, Claim 5.2.12 and Claim 5.2.13. □

Let $\mathcal{E}^\star$ denote the event that the random walk taken by our algorithm ends at a terminal meta-node at a depth $\leq \ell + 1$. It is trivial to note that $\Pr\left[\mathcal{E}^\star | \mathcal{E}^\star_{\texttt{t}}\right] = 1$. Henceforth, we condition on the event $\mathcal{E}^\star_{\texttt{cc-cont}}$. This means that the concerned random walk has reached some contaminated and congenitally clean meta-node x (say) at depth $\leq \ell$. The meta-node x has L children in $\mathcal{T}$, and by Lemma 5.2.4 at least $1/(40\ell)$-fraction of its children are terminal. Thus, with probability at least $1/(40\ell)$, in the very next step the random walk moves on to a terminal child of x, at depth $\leq \ell + 1$.

To summarize, we deduce that $\Pr[\mathcal{E}^\star | \mathcal{E}^\star_{\mathtt{t}} \cup \mathcal{E}^\star_{\mathtt{cc\text{-}cont}}] \geq 1/(40\ell)$. Lemma 5.2.2 now follows from (5.1) and Corollary 5.2.14.

5.2.5 Getting Rid of Assumption 5.2.1: The Major Technical Hurdles

If we remove Assumption 5.2.1, our algorithm might now fail if, given an uncolored edge $e = (u, v)$, it finds a sufficiently short alternating path P starting at u and ending at v. In this case, our algorithm runs $\mathsf{Apply}(\chi, e, P)$, but since P has both u *and* v as endpoints, this does not produce a proper coloring. In order to deal with this case, we need to use *Vizing fans* in order to construct this alternating path P. In addition to making the algorithm more technical, this leads to the following two major hurdles that we need to overcome.

Hurdle 1. Let x and y be two distinct congenitally clean meta-nodes with the same transcript $(\tau_0, \ldots, \tau_i)$. Previously, by Claim 5.2.6, we had that the paths $P^{(x)}_{\leq L}$ and $P^{(y)}_{\leq L}$ were vertex-disjoint. However, *this is no longer necessarily the case.* Every time we construct a fan around some vertex u, we make changes to the colors of the edges around u, even if they have colors that are not contained in any of the types of the alternating paths that have been used so far. Thus, the alternating path $P^{(x)}$ of some congenitally clean meta-node x might not be a maximal alternating path in the original coloring $\chi^{(r)}$. We say that such a meta-node x is *damaged*. Since each fan only changes the colors of at most Δ edges, each of which is contained in at most $O(\Delta)$ alternating paths, and our algorithm only runs for $\tilde{O}(1)$ steps, we can argue that each meta-node has at most $\tilde{O}(\Delta^2)$ many damaged children. By taking L to be sufficiently large, we can ensure that (with probability $\Omega(1)$) we do not encounter any damaged meta-nodes in a random walk (see Lemma 9.2.5).

Hurdle 2. Let $e = (u, v)$ be an uncolored edge and $c_u \in \mathsf{miss}_\chi(u)$, $c_v \in \mathsf{miss}_\chi(v)$ be blocking colors. Previously, our algorithm always found an alternating path P starting at u that did not use either c_u or c_v. The ability to find an alternating path that can avoid such blocking colors is crucial for the proof of Lemma 5.2.4. However, if we use Vizing fans to find alternating paths, then we can no longer guarantee that we can avoid using these blocking colors. In the case that we cannot avoid these blocking colors, we use a modified Vizing fan construction to find a $\{c_u, c_v\}$-alternating path that does not start at either u or v (see Lemma 9.1.2). Similarly to Lemma 5.2.4, we consider a different case where an $\Omega(1/\ell)$-fraction of the children of a meta-node have this property. In this situation, we can guarantee that $\Omega(L/\ell)$ of the alternating paths corresponding to these children must be vertex-disjoint, and thus have an average length of $\tilde{O}(\sqrt{\Delta n})$, leading to $\Omega(L/\ell)$ many of these children being terminal.

In Chapter 9, we give the complete proof of Theorem 5.1.2 without Assumption 5.2.1.

5.3 Dynamic Edge Coloring via the Nibble Method

In this section, we provide an overivew of our algorithms for $(1+\epsilon)\Delta$-coloring based on the Nibble method. We summarize our dynamic $(1+\epsilon)\Delta$-coloring algorithm in Theorem 1.1.8, which we restate

below. We give the full details of this algorithm in Chapter 10.

Theorem 1.1.8. *There is an algorithm that, given a dynamic graph G with maximum degree at most $\Delta \geq (\log n/\epsilon)^{\text{poly}(1/\epsilon)}$, maintains a $(1+\epsilon)\Delta$-coloring of G with $\text{poly}(1/\epsilon)$ expected worst-case update time, against an oblivious adversary.*

As a corollary of this result, we obtain our static $(1+\epsilon)\Delta$-coloring algorithm, which we summarize in Corollary 1.1.9.

Corollary 1.1.9. *There is an algorithm that, given a graph G with maximum degree at most $\Delta \geq (\log n/\epsilon)^{\text{poly}(1/\epsilon)}$, computes a $(1+\epsilon)\Delta$-coloring of G in $O(m\,\text{poly}(1/\epsilon))$ time with high probability.*

5.3.1 Perspective: The Quest for Constant Update Time

Achieving constant update times for fundamental problems is an important research agenda within dynamic algorithms [AS21, BCH17, BGK+22, BGM17, BHNW21, BK19, HP20, PS16, Sol16, SW18]. There are two major considerations that underpin this research agenda. (i) A constant update time algorithm rules out the possibility of obtaining a (cell-probe) lower bound for the concerned problem [Lar12, PD06]. (ii) It immediately implies a *linear time* algorithm for the concerned problem in the static setting,[6] and thus aligns with what is essentially the best possible static guarantee. With this backdrop, we encounter a significant hurdle at the very beginning of our quest, since currently there does not even exist a $O_\epsilon(m)$ time *static* algorithm for $(1+\epsilon)\Delta$-edge coloring.[7] In fact, if we insist upon getting an exact linear (i.e., $O(m)$) running time, and subject to this constraint try to minimize the number of colors being used, then the only game in town happens to be a very simple, folklore algorithm that gives us $(2+\epsilon)\Delta$-coloring. This algorithm can also be dynamized to get $O(1/\epsilon)$ update time, as explained below.

A Folklore (Randomized) Dynamic Algorithm. Suppose that we have a palette $\mathcal{C}$ of $(2+\epsilon)\Delta$ colors, and we are currently maintaining a proper coloring $\chi : E \longrightarrow \mathcal{C}$ of the input graph $G = (V, E)$. For each node $v \in V$, we maintain the set $\overline{P(v)} := \{c \in \mathcal{C} : \exists (u, v) \in E \text{ s.t. } \chi(u, v) = c\}$ of colors that are currently assigned to the edges incident on v, as a hash table. If an edge e gets deleted from G, then we don't do anything else as the coloring χ continues to remain proper. In contrast, if an edge (u, v) gets inserted into G, then we keep sampling colors u.a.r. from $\mathcal{C}$ until we find a *free* color $c \in \mathcal{C} \setminus \left(\overline{P(u)} \cup \overline{P(v)}\right)$ for this edge, and then we set $\chi(u, v) := c$ and update the hash tables $\overline{P(u)}, \overline{P(v)}$, which takes $O(1)$ expected time. Note that $\left|\overline{P(x)}\right| \leq \Delta$ for each endpoint $x \in \{u, v\}$, and hence there are at least $(2+\epsilon)\Delta - 2\Delta = \epsilon\Delta$ free colors for (u, v) when the edge gets inserted. Accordingly, in expectation we need to sample at most $|\mathcal{C}|/(\epsilon\Delta) = O(1/\epsilon)$ colors from $\mathcal{C}$ until we find a free color for (u, v). Furthermore, for each sampled color c', using the hash tables $\overline{P(u)}, \overline{P(v)}$ we can determine in $O(1)$ expected time whether or not c' is free. Putting everything together, this leads to a dynamic $(2+\epsilon)\Delta$-edge coloring algorithm with $O(1/\epsilon)$ expected update time, which can easily be converted into a static $(2+\epsilon)\Delta$-edge coloring algorithm with $O(m/\epsilon)$ expected run time.

[6] We take the (static) input graph, and feed it to the dynamic algorithm by inserting its edges one at a time.
[7] We use the notation $O_\epsilon(\cdot)$ to hide $\text{poly}(1/\epsilon)$ factors.

Existing Barriers. At this point, we revisit the state-of-the-art on dynamic $(1+\epsilon)\Delta$-edge coloring, and explain the challenges behind extending the known techniques to obtain constant update time.

(I) The two known dynamic algorithms for $(1+\epsilon)\Delta$-edge coloring [DHZ19, Chr23] are both analyzed in a *memory-less* manner. Specifically, they assume that we start with any arbitrary, adversarially chosen $(1+\epsilon)\Delta$-edge coloring in the current graph G, and then show how to modify that coloring (to ensure that it remains proper) in polylogarithmic time after the insertion/deletion of an edge. A lower bound construction from [CHL+20], however, implies that any such memory-less analysis must necessarily imply an update time of $\Omega(\log(\epsilon n)/\epsilon)$.

(II) There is a weaker version of the dynamic edge coloring problem, where we care about the *recourse* (as opposed to update time) of the maintained solution, which basically equals the number of changes the algorithm makes to the coloring after an update. There exists a $(1+\epsilon)\Delta$-edge coloring algorithm with $O_\epsilon(1)$ recourse [BGW21], based on the NIBBLE method (see Section 5.3.2 for more details) that was first used in the context of edge coloring in the distributed setting [DGP98]. It seems very difficult to implement the algorithm of [BGW21] using $O_\epsilon(1)$ update time data structures, for two reasons. First, the [BGW21] dynamic algorithm needs to resample the color c of an edge $e = (u, v)$ when its palette $P(e) = \mathcal{C} \setminus \left(\overline{P(u)} \cup \overline{P(v)}\right)$ changes by a small amount, even if c continues to be part of $P(e)$. It is not at all clear how to implement this resampling efficiently, i.e., in constant update time. Second, and more fundamentally, all existing Nibble method-based algorithms require some form of *regularization gadget*, since the inductive approach that is used to analyze these algorithms does not work on graphs that are not near-regular. Implementing this gadget requires $\Omega(n\Delta)$ running time in the static setting, and $\Omega(n\Delta)$ preprocessing time in the dynamic setting.

5.3.2 Our Results

We are now ready to present our results. Towards this end, we define the following parameter

$$\Delta^\star := \left(\log n/\epsilon^4\right)^{\Theta((1/\epsilon)\log(1/\epsilon))}.$$

Recall that n and m respectively denote the number of nodes and edges in the input graph $G = (V, E)$, and let Δ be an upper bound on the maximum degree of G.

Theorem 5.3.1. *In the static setting, we can compute a $(1+\epsilon)\Delta$-edge coloring in the input graph G in $O(m\log(1/\epsilon)/\epsilon^2)$ time w.h.p., provided $\Delta \geq \Delta^\star$.*

We then extend our algorithm to the dynamic setting, and derive the theorem below.

Theorem 5.3.2. *We can maintain a $(1+\epsilon)\Delta$-edge coloring in a dynamic graph G in $O(\log^4(1/\epsilon)/\epsilon^9)$ expected worst-case update time (against an oblivious adversary), provided $\Delta \geq \Delta^\star$.*

We can convert the expected worst-case update time bound of Theorem 5.3.2 into a high probability amortized update time guarantee, for polynomially long update sequences (see Corollary 10.3.6).

5.3.3 Our Techniques

A one-sentence summary of our approach is that we combine the NIBBLE algorithm with the subsampling technique used by [KLS+22] in the context of online edge coloring, and then always maintain the precise output of the resulting static algorithm as the input graph undergoes edge insertions/deletions. We now explain this idea in more detail.

Our Static Algorithm. We start by summarizing the NIBBLE algorithm (see Section 5.4.1). It runs in $T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ rounds, on the input graph $G = (V, E)$ with a palette $\mathcal{C}$ of $(1+\epsilon)\Delta$ colors. At the start of round $i \in [T]$, let $P_i(v)$ denote the palette of a node $v \in V$, which consists of all the colors that have *not* yet been (tentatively) assigned to any edge incident on v. In round i, each uncolored edge e *selects* itself independently with probability ϵ. Next, every selected edge $e = (u, v)$ picks a tentative color $\tilde{\chi}(e)$ independently and u.a.r. from its palette $P_i(u) \cap P_i(v)$. At the end of T rounds, we collect all the *failed* edges $F \subseteq E$; these are the edges that were either not selected during any of the T rounds and hence did not receive any tentative color, or received a tentative color which conflicts with one of its neighbors. We now color the subgraph $G_F := (V, F)$, using the folklore algorithm and an extra palette of $O(\Delta(G_F))$ colors that is mutually disjoint with $\mathcal{C}$. This, combined with the tentative colors assigned to the edges in $E \setminus F$, gives us a proper $(1+\epsilon)\Delta + O(\Delta(G_F))$-coloring of G. The main challenge now is to show that $\Delta(G_F) = O(\epsilon\Delta)$ w.h.p., for that would give us a $(1 + O(\epsilon))\Delta$-coloring of G.

Next, we observe that we do not need any regularizing gadget to analyze the NIBBLE algorithm, *if the input graph G is a forest* (see Section 5.4.2). This is primarily because under such a scenario, just before we pick a tentative color for an edge $(u, v) \in E$ in some round $i \in [T]$, the palettes $P_i(u)$ and $P_i(v)$ are mutually independent. This observation makes it easy to obtain a $O_\epsilon(m)$ time implementation of the NIBBLE algorithm for $(1+\epsilon)\Delta$-edge coloring. We essentially use the same hash table data structures which allow us to efficiently implement the folklore algorithm (see Section 5.3.1), along with the fact that w.h.p., at the start of each round $i \in [T]$, the palette $P_i(u, v) := P_i(u) \cap P_i(v)$ of each uncolored edge is of size $\Omega(\epsilon^2\Delta)$ (see Corollary 5.4.8).

At this point, we move on to the general case where the input graph G might contain cycles (see Section 5.4.3). Here, we first observe that the palette $P_i(v)$ of a node v for a round $i \in [T]$ depends only on the i-hop neighborhood of v. Say that a node v is *good* in G if its $(T+1)$-hop neighborhood in G does not contain any cycle, and *bad* otherwise. Since the NIBBLE algorithm runs for only T rounds, we can simply pretend that the input graph is a forest while analyzing what the algorithm does to a good node and all its incident edges (see Lemma 5.4.17). In particular, we can show that w.h.p. every good node will have degree at most $O(\epsilon\Delta)$ in G_F.

We now combine the previous observations with a subsampling technique [KLS+22] (see Section 5.4.4). The basic idea is simple. Fix two parameters $\gamma := 1/(30T)$ and $\Delta' := \Delta^\gamma$, and set $\eta := \Delta/\Delta'$. Next, partition the input graph G into η subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$, by placing each edge $e \in E$ independently and u.a.r. in one of the subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$. This partition happens to satisfy the following two properties. (I) W.h.p. $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta'$ for all $j \in [\eta]$. (II) Say that an edge $e = (u, v) \in E$ is *problematic* iff either u or v is a bad node in $\mathcal{G}_j$, where $j \in [\eta]$ is the unique index

such that $e \in \mathcal{G}_j$. Let $E^\star \subseteq E$ be the set of all problematic edges, and let $G^\star := (V, E^\star)$. Then $\Delta(G^\star) = O(\epsilon\Delta)$ w.h.p. Armed with these two observations, our final algorithm on general graphs works as follows. We compute the partition of the input graph G into the subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$. For each $j \in [\eta]$, we run the NIBBLE algorithm on $\mathcal{G}_j$ in an attempt to color it with a (distinct) palette $\mathcal{C}_j$ of $(1 + O(\epsilon))\Delta'$ colors. Overall, this requires $\eta \cdot (1 + O(\epsilon))\Delta' = (1 + O(\epsilon))\Delta$ colors. At the end of this step, we are left with two types of failed edges that could not be properly colored: the ones that are problematic, and the ones that are not. Because of the locality of the NIBBLE method, for each $j \in [\eta]$, w.h.p. each node $v \in V$ is incident on at most $O(\epsilon\Delta')$ non-problematic failed edges in $\mathcal{G}_j$. Thus, w.h.p. the maximum degree of the subgraph consisting of all non-problematic edges over all $j \in [\eta]$ is at most $\eta \cdot O(\epsilon\Delta') = O(\epsilon\Delta)$. Next, by Property II above, the maximum degree of the subgraph consisting of all problematic edges, over all $j \in [\eta]$, is given by $\Delta(G^\star) = O(\epsilon\Delta)$. Putting everything together, we infer that at the end of the first step, the subgraph consisting of all failed edges has maximum degree $O(\epsilon\Delta)$. Hence, we can easily color these remaining failed edges, using the folklore algorithm from Section 5.3.1 and an extra set of $O(\epsilon\Delta)$ colors. This leads to a $(1 + O(\epsilon))\Delta$-coloring of the input graph G, without any additional overhead in the running time compared to the scenario where G was a forest, because the subsampling step can easily be implemented very efficiently.

Our Dynamic Algorithm. We dynamize our static algorithm using a very natural approach (see Section 5.5). It is not surprising that the subsampling step is relatively easy to dynamize, so here we only focus on highlighting what our dynamic algorithm does when the input graph remains a forest. Essentially, our dynamic algorithm hinges upon two main observations.

(I) When an edge e gets inserted, we might as well fix an index $i_e \in [T + 1]$ by sampling i_e from a capped geometric distribution with probability ϵ,[8] and we can also fix an infinite length color-sequence c_e, such that for each $\ell \in \mathbb{Z}^+$ the ℓ^{th} entry $c_e(\ell)$ in this sequence is a color sampled independently and u.a.r. from the input palette $\mathcal{C}$. These rounds $\{i_e\}_e$ and color-sequences $\{c_e\}_e$ uniquely determine the output of the NIBBLE algorithm on a given input graph $G = (V, E)$. Specifically, each edge $e \in E$ selects itself in round i_e, and then identifies the smallest integer $\ell \in \mathbb{Z}^+$ such that $c_e(\ell) \in P_i(e)$, and sets $\tilde{\chi}(e) := c_e(\ell)$.[9] Throughout the sequence of updates, we simply maintain the output of this static algorithm w.r.t. the indices $\{i_e\}_e$ and color-sequences $\{c_e\}_e$. We show that this natural approach itself suffices to guarantee an expected worst-case recourse of $O_\epsilon(1)$ (see Section 5.5.1), and is in sharp contrast with the algorithm of [BGW21] which required repeated resampling of colors.

(II) We need one additional insight to implement our low-recourse algorithm in $O_\epsilon(1)$ update time. Specifically, we *truncate* the color-sequences $\{c_e\}$ at length $K := \Theta((1/\epsilon^2)\log(1/\epsilon))$, i.e., for each edge e, we stop constructing the sequence c_e after sampling the first K colors $c_e(1), \ldots, c_e(K)$. The intuition behind why everything still works is as follows. W.h.p., we know that $|P_{i_e}(e)| = \Omega(\epsilon^2\Delta)$. Thus, conditioned on this event, at least one of the first K colors in c_e should appear in $P_i(e)$ with

[8] Thus, we have $\Pr[i_e = i] = (1 - \epsilon)^{i-1}\epsilon$ for all $i \in [T]$ and $\Pr[i_e = T + 1] = (1 - \epsilon)^T$.

[9] Note that this is equivalent to sampling a color $\tilde{\chi}(e)$ from $\mathcal{C}$ u.a.r.

probability at least $1 - \epsilon$. In other words, the resulting algorithm is equivalent to the following process: We throw away each edge $e \in E$ with probability ϵ, and we run the NIBBLE algorithm on the surviving edges. The subgraph consisting of the edges that get thrown away has maximum degree $O(\epsilon\Delta)$ w.h.p., and hence we can separately color this subgraph using the folklore algorithm from Section 5.3.1. It turns out that we can develop data structures that support the implementation of this modified algorithm in $O_\epsilon(1)$ expected worst-case update time (see Section 5.5.2). The main reason is that the truncation step acts in a way which is reminiscent of *palette-sparsification* [DHZ19]. Indeed, now each edge gets assigned a tentative color that comes from a small set of size $K = O_\epsilon(1)$. This helps us design a supporting data structure whose expected update time is proportional to the recourse of the algorithm from step (I) above, which, we already know to be $O_\epsilon(1)$.

5.3.4 Remark on the Lower Bound on Δ

In the edge coloring literature, it is common to assume that $\Delta = \Omega(\text{polylog}(n))$ [BGW21, DHZ19, KLS+22]. The reason we need the lower bound on Δ is as follows (see Section 5.4.4 for details). We partition the input graph G into $\eta = \Delta/\Delta'$ subgraphs $G_1, \ldots, G_\eta$, by throwing each edge of G u.a.r. into one of these subgraphs. We now need to enforce the following two properties. (i) Δ' needs to be large enough to ensure that we have enough concentration to be able to run the Nibble algorithm on each G_i. (ii) Δ' needs to be small enough to ensure that each G_i is sufficiently "locally treelike" for our analysis to go through. In particular, so that the subgraph $G^\star$, which consists of all the bad edges, has maximum degree at most $\epsilon\Delta$ (see Claim 5.4.19), because these bad edges will get separately colored using a greedy algorithm. Now, it so happens that for Property (i) to hold, we need $\Delta' \geq \Omega(\log n/\epsilon^4)$, and for Property (ii) to hold, we need $\Delta' \leq \Delta^{\Theta(1/T)}$, where $T = \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ is the number of rounds required by the Nibble method. Combining these two inequalities together, we get that $\Delta \geq \left(\log n/\epsilon^4\right)^{\Theta((1/\epsilon)\log(1/\epsilon))}$. We leave it as a challenging open question to improve this lower bound on Δ.

5.4 Overview of our Static Nibble Algorithm

In this section, we describe how our algorithm (see Theorem 5.3.1) works in the static setting, and present an overview of the key ideas that underpin its analysis. To convey the main intuition behind our framework, here we intentionally explain some of the arguments in an informal/semi-rigorous manner. The complete, formal proofs from this section are deferred to Section 10.1 and Section 10.4.

Organization. In Section 5.4.1, we present the NIBBLE algorithm. Section 5.4.2 explains how to analyze and implement this algorithm when the input graph is a forest. In Section 5.4.3, we show that on general graphs, the analysis from Section 5.4.2 still holds for all those nodes that are *not* part of any short cycle. Finally, Section 5.4.4 contains an overview of our final algorithm, which involves combining the NIBBLE method along with a subsampling technique of [KLS+22].

5.4.1 The NIBBLE Algorithm

Fix any input graph $G = (V, E)$ with n nodes and maximum degree at most Δ, any constant $\epsilon \in (0, 1/10)$, and any *palette* $\mathcal{C}$ of $\lceil(1+\epsilon)\Delta\rceil$ colors. The NIBBLE algorithm runs for $T := \lfloor(1/\epsilon)\log(1/\epsilon)\rfloor$ *rounds*. At the start of round $i \in [T]$, we have a subset of edges $E_i \subseteq E$ such that the algorithm has already assigned tentative colors to the remaining edges $E \setminus E_i$. We denote this *tentative* partial coloring by $\tilde{\chi} : E \setminus E_i \to \mathcal{C} \cup \{\perp\}$, which need not necessarily be proper. For each node $v \in V$, we refer to the set of colors $P_i(v) := \mathcal{C} \setminus \tilde{\chi}(N(v) \setminus E_i)$ as the *palette* of v at the start of round i, where $N(v) \subseteq E$ is the set of edges incident on v in G. In words, the palette $P_i(v)$ consists of the set of colors that were *not* tentatively assigned to any incident edge of v in previous rounds. We define $P_i(u, v) := P_i(u) \cap P_i(v)$ to be the *palette* of any edge $(u, v) \in E_i$ at the start of round i.

We start by initializing $E_1 \leftarrow E$, and $P_1(v) \leftarrow \mathcal{C}$ for all $v \in V$. Subsequently, for $i = 1, \ldots, T$, we implement round i as follows. Each edge $e \in E_i$ *selects* itself independently with probability ϵ. Let $S_i \subseteq E_i$ be the set of selected edges. Next, in parallel, each edge $e \in S_i$ samples a color $\tilde{\chi}(e)$ independently and uniformly at random from $P_i(e)$. For any edge $e \in S_i$, if $P_i(e) = \emptyset$ then we set $\tilde{\chi}(e) \leftarrow \perp$. At this point, we define the collection $F_i \subseteq S_i$ of *failed* edges in round i. We say that an edge $e = (u, v) \in S_i$ *fails* in round i iff either (i) $\tilde{\chi}(e) = \perp$, or (ii) there is a neighboring edge $f \in (N(u) \cup N(v)) \cap S_i$ which was also selected in round i and received the same tentative color as the edge e (i.e., $\tilde{\chi}(e) = \tilde{\chi}(f)$). Let $F_i \subseteq S_i$ denote this collection of failed edges (in round i). We now set $E_{i+1} \leftarrow E_i \setminus S_i$ and proceed to the next round $i + 1$.

To ease notations, at the end of the last round T we define $F_{T+1} \leftarrow E_{T+1}$, and $\tilde{\chi}(e) \leftarrow \perp$ for all $e \in F_{T+1}$. We let $F := \bigcup_{i=1}^{T+1} F_i$ denote the set of failed edges across all the rounds. It is easy to check that the tentative coloring $\tilde{\chi}$, when restricted to the edge-set $E \setminus F$, is already proper.

Observation 5.4.1. *$\tilde{\chi}$ is a proper $(1+\epsilon)\Delta$-edge coloring in the subgraph $G_{E\setminus F} := (V, E \setminus F)$.*

The pseudocode of this procedure appears in Algorithm 6. We now describe some further notations that will be used throughout the rest of this chapter.

Notations. For all $v \in V$ and $i \in T$, we define $N_i(v) := N(v) \cap S_i$ to be the set of edges incident on the node v that get selected in round i. For each edge $e = (u, v) \in E$, let $N(e) := N(u) \cup N(v)$ denote the set of its neighboring edges. Furthermore, for any graph $H = (V, E_H)$ and any node $v \in V$, let $\deg_H(v)$ denote the *degree* of v in H, and let $\Delta(H)$ denote the *maximum degree* of any node in H. We will sometimes abuse these notations and write $\deg_{E_H}(v)$ and $\Delta(E_H)$ when the node-set V is clear from the context. Finally, given any sequence of sets $A_1, A_2, \ldots$, we will use the shorthands $A_{<i} := \bigcup_{j<i} A_j$, $A_{\leq i} := \bigcup_{j\leq i} A_j$, $A_{>i} := \bigcup_{j>i} A_j$ and $A_{\geq i} := \bigcup_{j\geq i} A_j$. For instance, this means that $S_{<i} := \bigcup_{j=1}^{i-1} S_j$ (see Line 7 in Algorithm 6).

Intuitively, it is easy to see that the NIBBLE algorithm is *symmetric* w.r.t. the palette $\mathcal{C}$, i.e., it does not give preference to one color over another. The lemma below formalizes this intuition, and will be repeatedly invoked during our analysis (we defer its proof to Section 10.1.2).

Algorithm 6: NIBBLE$(G = (V, E), \Delta, \epsilon)$

$\mathcal{C} \leftarrow [(1+\epsilon)\Delta]$, $E_1 \leftarrow E$, and $\tilde{\chi}(e) \leftarrow \perp$ for all $e \in E$
for $i = 1, ..., T$ **do**
  $S_i \leftarrow \emptyset$
  for $e \in E_i$ **do**
    Add e to S_i independently with probability ϵ
  for $e = (u, v) \in S_i$ **do**
    $P_i(e) \leftarrow \mathcal{C} \setminus \tilde{\chi}(N(e) \cap S_{<i})$
    if $P_i(e) \neq \emptyset$ **then**
      Sample $\tilde{\chi}(e) \sim P_i(e)$ independently and u.a.r.
  $F_i \leftarrow \{e \in S_i \mid \exists f \in N(e) \cap S_i \text{ such that } \tilde{\chi}(f) = \tilde{\chi}(e)\} \cup \{e \in S_i \mid \tilde{\chi}(e) = \perp\}$
  $E_{i+1} \leftarrow E_i \setminus S_i$
$F_{T+1} \leftarrow E_{T+1}$
$F \leftarrow \bigcup_{i=1}^{T+1} F_i$
return $\tilde{\chi}, F$

Lemma 5.4.2. *Fix the random bits used by the* NIBBLE *algorithm that determine which edges get selected in which rounds. Then for all $u \in V$, $i \in [T]$, $C \subseteq \mathcal{C}$ and permutations $\pi : \mathcal{C} \longrightarrow \mathcal{C}$, we have*

$$\Pr[P_i(u) = C] = \Pr[\pi(P_i(u)) = C].$$

5.4.2 Analysis of the NIBBLE Algorithm on Forests

The NIBBLE algorithm returns a proper $(1+\epsilon)\Delta$-edge coloring $\tilde{\chi}$ on the subgraph $G_{E\setminus F} = (V, E \setminus F)$ (see Observation 5.4.1). We now wish to argue that the remaining subgraph $G_F := (V, F)$ has small maximum degree. Specifically, if it so happens that $\Delta(G_F) = O(\epsilon\Delta)$, then we can just color the edges of G_F, using the folklore algorithm from Section 5.3.1 and an extra set of $O(\epsilon\Delta)$ colors. This, combined with the coloring $\tilde{\chi}$ of $G_{E\setminus F}$, would give us a $(1 + O(\epsilon))\Delta$-coloring of G. We now prove this desired upper bound on $\Delta(G_F)$, under the assumption that the input graph G is a forest with sufficiently large maximum degree. The main result in this section is summarized below.

Lemma 5.4.3. *Let G be a forest with maximum degree $\Delta \geq \frac{(100 \log n)}{\epsilon^4}$. Then w.h.p. we have:*

$$deg_F(v) = O(\epsilon\Delta) \text{ for all nodes } v \in V.$$

We start with the following key observation.

Observation 5.4.4. *Let $G = (V, E)$ be a forest. Fix the random bits used by the* NIBBLE *algorithm that determine which edges get selected in which rounds. Consider any $i \in [T]$, $u \in V$, and edges $(u, v_1), \ldots, (u, v_k) \in S_i$. Then the palettes $\{P_i(u), P_i(v_1), \ldots, P_i(v_k)\}$ are mutually independent.*

Proof Sketch. The palettes $P_i(u), P_i(v_1), \ldots, P_i(v_k)$ only depend on what happens during rounds $< i$ in $G_{<i} := (V, S_{<i})$, and the nodes $u, v_1, \ldots, v_k$ lie in different connected components of $G_{<i}$. □

In Lemma 5.4.5, Corollary 5.4.6 and Lemma 5.4.7, we derive concentration bounds on the sizes of the sets $N_i(v)$ and the palettes of nodes/edges at the start of each round. Later on, we use these concentration bounds in Section 5.4.2, which gives an overview of the proof of Lemma 5.4.3. Finally, we explain how to efficiently implement the NIBBLE algorithm in linear time in Section 5.4.2.

Lemma 5.4.5. *Let $\mathcal{Z}$ denote the event which occurs iff we have $|N_i(u)| < (1+\epsilon)\cdot\epsilon(1-\epsilon)^{i-1}\Delta$ for all nodes $u \in V$ and all rounds $i \in [T]$. Then the event $\mathcal{Z}$ occurs w.h.p.*

Proof Sketch. Fix any node $u \in V$ and any round $i \in [T]$. Any given edge $(u,v) \in E$ incident on u appears in S_i with probability $\epsilon(1-\epsilon)^{i-1}$. Thus, by linearity of expectation, we have $\mathbb{E}\left[|N(u) \cap S_i|\right] \leq \epsilon(1-\epsilon)^{i-1}\Delta$. Furthermore, note that each edge decides independently (of all other edges) whether to get selected in round i. Since $\Delta \geq \frac{(100\log n)}{\epsilon^4}$ and $T = \lfloor (1/\epsilon)\log(1/\epsilon)\rfloor$, the lemma now follows from an application of Chernoff bound, and a union bound over all $u \in V, i \in [T]$. □

Corollary 5.4.6. *Conditioned on $\mathcal{Z}$, we have $|P_i(u)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta$ for all $u \in V, i \in [T]$.*

Proof. Fix any node $u \in V$ and any round $i \in [T]$. Conditioned on the event $\mathcal{Z}$, we have:

$$\left|\bigcup_{j=1}^{i-1} N_j(u)\right| < \sum_{j=1}^{i-1}(1+\epsilon)\cdot\epsilon(1-\epsilon)^{j-1}\Delta = (1+\epsilon)\Delta\cdot\left(1-(1-\epsilon)^{i-1}\right). \tag{5.5}$$

The corollary now follows from (5.5) and the observation that $|P_i(u)| \geq (1+\epsilon)\Delta - \left|\bigcup_{j=1}^{i-1} N_j(u)\right|$. □

Lemma 5.4.7. *Fix the random bits used by the* NIBBLE *algorithm that determine which edges are selected in which rounds, in* any *way which ensures that the event $\mathcal{Z}$ occurs. Then w.h.p., we have $|P_i(e)| > (1-\epsilon^2)(1-\epsilon)^{2(i-1)}\Delta$ for all rounds $i \in [T]$ and all edges $e \in E_i$.*

Proof Sketch. Consider any $i \in [T]$ and any edge $e = (u,v) \in E_i$. For each color $c \in \mathcal{C}$ and each node $w \in V$, let $X_c^w \in \{0,1\}$ be an indicator random variable that is set to 1 iff $c \in P_i(w)$. Clearly, we have $|P_i(e)| = \sum_{c\in\mathcal{C}} X_c^u \cdot X_c^v$. By Observation 5.4.4, X_c^u and X_c^v are independent, and hence:

$$\mathbb{E}\left[|P_i(e)|\right] = \sum_{c\in\mathcal{C}} \mathbb{E}\left[X_c^u \cdot X_c^v\right] = \sum_{c\in\mathcal{C}} \mathbb{E}\left[X_c^u\right]\cdot\mathbb{E}\left[X_c^v\right]. \tag{5.6}$$

We now consider any fixed color $c \in \mathcal{C}$, and derive a lower bound on $\mathbb{E}[X_c^u]$.[10] Using the symmetry of the NIBBLE algorithm w.r.t. the colors (see Lemma 5.4.2), in conjunction with the fact that $|P_i(u)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta$ (see Lemma 5.4.5), we get:

$$\mathbb{E}\left[X_c^u\right] = \Pr\left[X_c^u = 1\right] \geq \frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{|\mathcal{C}|}. \tag{5.7}$$

[10] A lower bound on $\mathbb{E}[X_c^v]$ can be derived in the same way.

Since $|\mathcal{C}| = (1+\epsilon)\Delta$, from Equation (5.6) and Equation (5.7), we derive that:

$$\mathbb{E}\left[|P_i(e)|\right] \geq |\mathcal{C}| \cdot \left(\frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{|\mathcal{C}|}\right)^2 = (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta. \tag{5.8}$$

With some extra effort, we can modify the above argument by working with a slightly different (but analogous) set of random variables Y_c^w (instead of X_c^w) such that the collection of random variables $\{Y_c^u \cdot Y_c^v\}_{c \in \mathcal{C}}$ is *negatively associated* (see Definition 10.5.5). This allows us to derive a concentration bound out of Equation (5.8), which leads to the proof of the lemma (see Section 10.1 for details). □

Corollary 5.4.8. *Fix the random bits used by the* Nibble *algorithm that determine which edges are selected in which rounds, in such a way that the event $\mathcal{Z}$ occurs. Then w.h.p., we have $|P_i(e)| \geq \epsilon^2(1+\epsilon)\Delta/8$ for all rounds $i \in [T]$ and all edges $e \in E_i$.*

Proof Sketch. Follows from Lemma 5.4.7 and the observation that $i \leq T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$. □

Corollary 5.4.8 will be useful in analyzing the runtime of the Nibble algorithm in Section 5.4.2.

Proof (Sketch) of Lemma 5.4.3

Consider any node $v \in V$, and let $N_F(v) = N(v) \cap F$ be the set of *failed edges* incident on v. We will show that $\deg_F(v) = |N_F(v)| = O(\epsilon\Delta)$ w.h.p. Lemma 5.4.3 will then follow from a union bound over all nodes $v \in V$. We begin by partitioning the set $N_F(v)$ into three subsets:

- $N_F^{(0)}(v) := N_F(v) \cap F_{T+1}$ (see Line 12 in Algorithm 6)
- $N_F^{(1)}(v) := \left\{e \in N_F(v) \setminus N_F^{(0)}(v) : \tilde{\chi}(e) = \perp\right\}$
- $N_F^{(2)}(v) := N_F(v) \setminus \left(N_F^{(0)}(v) \cup N_F^{(1)}(v)\right)$

We refer to the edges in $N_F^{(0)}(v)$ as *residual edges*, since they are not selected in any round $i \in [T]$. The set $N_F^{(1)}(v)$ consists of those edges $e \in N_F(v)$ which got selected in some round $i \in [T]$ but unfortunately had $P_i(e) = \emptyset$. Finally, the set $N_F^{(2)}(v)$ consists of the remaining edges in $N_F(v)$, the ones who received the same tentative color as (at least one of) their neighbors in some round.

We will separately upper bound the sizes of the sets $N_F^{(0)}(v)$, $N_F^{(1)}(v)$ and $N_F^{(2)}(v)$.

Claim 5.4.9. *We have $\left|N_F^{(0)}(v)\right| = O(\epsilon\Delta)$ w.h.p.*

Proof Sketch. Any given edge $(u,v) \in E$ appears in the set $F_{T+1} = E \setminus S_{<T+1}$ with probability $= (1-\epsilon)^T \leq e^{-\epsilon T} = \epsilon$. This implies that $\mathbb{E}\left[\left|N_F^{(0)}(v)\right|\right] = O(\epsilon\Delta)$. Since $\Delta = \Omega\left(\log n/\epsilon^4\right)$ and since the random bits that determine whether different edges appear in F_{T+1} are independent of each other, the proof now follows from an application of Chernoff bounds. □

For the rest of Section 5.4.2, we fix the random bits used by the NIBBLE algorithm that determine which edges are selected in which rounds, in such a way that the event $\mathcal{Z}$ occurs (the event $\mathcal{Z}$ occurs w.h.p., according to Lemma 5.4.5). Lemma 5.4.3 will follow from Claim 5.4.9, Claim 5.4.10 and Corollary 5.4.13.

Claim 5.4.10. *We have* $\left|N_F^{(1)}(v)\right| = 0$ *w.h.p.*

Proof Sketch. Lemma 5.4.7 implies that w.h.p. the following event occurs: $P_i(e) \neq \emptyset$ for all $i \in [T]$ and all $e \in S_i$. Conditioned on this event, no edge gets added to the set $N_F^{(1)}(v)$. □

We now focus on deriving an upper bound on $\left|N_F^{(2)}(v)\right|$. We start with the following claim.

Claim 5.4.11. *Consider any round* $i \in [T]$*, any edge* $(u, v) \in S_i$*, and any endpoint* $x \in \{u, v\}$*. Then we have:* $\Pr\left[\ \exists\ e \in N_i(x) \setminus \{(u, v)\} : \tilde{\chi}(u, v) = \tilde{\chi}(e)\right] \leq \epsilon$.

Proof Sketch. W.l.o.g. suppose that $x = u$ (the proof for the case $x = v$ is symmetric). Consider any edge $(u, w) \in S_i$ with $w \neq v$. We first lower bound the probability that $\tilde{\chi}(u, v) = \tilde{\chi}(u, w)$.

Observation 5.4.4 implies that the palettes $\{P_i(u), P_i(v), P_i(w)\}$ are mutually independent. Furthermore, since we have conditioned on the event $\mathcal{Z}$, Corollary 5.4.6 lower bounds the sizes of each of these palettes $\{P_i(u), P_i(v), P_i(w)\}$. Now, fix (i.e., condition on) the palettes $P_i(u)$ and $P_i(v)$, and let $c = \tilde{\chi}(u, v) \in P_i(u) \cap P_i(v)$ be the tentative color received by the edge (u, v) in round i. Using the symmetry of the NIBBLE algorithm w.r.t. the colors (see Lemma 5.4.2), and recalling that $|P_i(u)| \geq (1 + \epsilon)(1 - \epsilon)^{i-1}\Delta$ as per Corollary 5.4.6, we get:

$$\Pr[\tilde{\chi}(u, w) = c] \ \leq\ \frac{1}{|P_i(u)|}\ \leq\ \frac{1}{(1 + \epsilon)(1 - \epsilon)^{i-1}\Delta}.$$

Therefore, we conclude that $\Pr\left[\tilde{\chi}(u, v) \neq \tilde{\chi}(u, w)\right] \geq \left(1 - \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}\right)$, and hence:

$$\Pr\left[\tilde{\chi}(u, v) \neq \tilde{\chi}(u, w)\ \forall\ (u, w) \in N_i(u) \setminus \{(u, v)\}\right] \geq \left(1 - \frac{1}{(1 + \epsilon)(1 - \epsilon)^{i-1}\Delta}\right)^{|N_i(u)|} \geq 1 - \epsilon. \quad (5.9)$$

The first inequality holds because of Observation 5.4.4, whereas the last inequality holds since we have conditioned on the event $\mathcal{Z}$ (see Lemma 5.4.5). The lemma follows if we consider the probability of the complement of the event captured by Equation (5.9). □

Claim 5.4.12. *Consider any round* $i \in [T]$*. Then* $\left|N_F^{(2)}(v) \cap S_i\right| = O(\epsilon^2(1 - \epsilon)^{i-1}\Delta)$ *w.h.p.*

Proof Sketch. Consider the following two subsets of edges incident on v.

- $F_i'(v) := \{(u, v) \in S_i : \exists (v, w) \in S_i$ with $\tilde{\chi}(u, v) = \tilde{\chi}(v, w)\}$
- $F_i''(v) := \{(u, v) \in S_i : \exists (u, w) \in S_i$ with $\tilde{\chi}(u, v) = \tilde{\chi}(u, w)\}$

Observe that $N_F^{(2)}(v) \subseteq F_i'(v) \cup F_i''(v)$. Now, consider any edge $(u, v) \in N_i(v)$. By Claim 5.4.11, we have $\Pr[(u, v) \in F_i'(v)] \leq \epsilon$ and $\Pr[(u, v) \in F_i''(v)] \leq \epsilon$. Thus, by linearity of expectation, we get:

$$\mathbb{E}\left[\left|N_F^{(2)}(v)\right|\right] \leq \mathbb{E}\left[\left|F_i'(v)\right|\right] + \mathbb{E}\left[\left|F_i''(v)\right|\right] \leq 2\epsilon \cdot |N_i(v)| = O(\epsilon^2(1-\epsilon)^{i-1}\Delta). \tag{5.10}$$

In the above derivation, the last step holds because we conditioned on the event $\mathcal{Z}$ (see Lemma 5.4.5). With a little bit of extra work, we can infer that: (a) $|F_i'(v)|$ can be expressed as a function of a collection of random variables that is Lipschitz with all constants 2 (see Definition 10.5.3), which allows us to derive a concentration bound on $|F_i'(v)|$ using the method of bounded differences (see Proposition 10.5.4). (b) Using the fact that the input graph G is a forest, $|F_i''(v)|$ can be expressed as the sum of mutually independent 0/1 random variables, which allows us to derive a concentration bound on $|F_i''(v)|$ by applying a Chernoff bound. This concludes the proof of the claim. □

Corollary 5.4.13. *We have:* $\left|N_F^{(2)}(v)\right| = O(\epsilon\Delta)$ *w.h.p.*

Proof. Since $N_F^{(2)}(v) \subseteq \bigcup_{i=1}^{T} S_i$, from Claim 5.4.12 we derive that whp:

$$\left|N_F^{(2)}(v)\right| = O\left(\sum_{i=1}^{T} \epsilon^2(1-\epsilon)^{i-1}\Delta\right) \;=\; O\left(\epsilon(1-(1-\epsilon)^T)\Delta\right) \;=\; O(\epsilon\Delta).$$

□

Lemma 5.4.3 now follows from Claim 5.4.9, Claim 5.4.10, Lemma 5.4.5 and Corollary 5.4.13.

Running Time of the Nibble Algorithm on Forests

We now briefly describe how we can implement the Nibble algorithm in linear time when the input graph $G = (V, E)$ is a forest. We begin by scanning through all of the edges $e \in E$, and assigning a round $i_e \in [T+1]$ to each edge e, sampled i.i.d. from a capped geometric distribution with success probability ϵ. We then construct the sets $S_1, \ldots, S_{T+1}$, where S_i consists of the edges that will be colored during round i. This can be done in $O(m)$ time. For each $i = 1, \ldots, T$, we then scan through all of the edges $e \in S_i$ and sample a color $\tilde{\chi}(e) \sim P_i(e)$ independently and u.a.r. In order to implement this sampling, suppose that we can check whether or not a given color c is contained in $P_i(e)$ in $O(1)$ time. Since we know by Corollary 5.4.8 that $|P_i(e)| = \Omega(\epsilon^2\Delta)$ w.h.p., we can sample a color c from $\mathcal{C}$ independently and u.a.r. and this color will be contained in $P_i(e)$ with probability $\Omega(\epsilon^2)$. Hence, in expectation, we need to sample $O(1/\epsilon^2)$ many colors before finding one that is contained in $P_i(e)$. It follows that the total expected time required to sample all of the tentative colors is $O(m/\epsilon^2)$. Finally, we can color all the failed edges using the folklore algorithm from Section 5.3.1 in $O(m)$ time.

It turns out that using hash tables we can maintain, for each node $v \in V$, the complement of its palette, given by $\overline{P(v)} := \{c \in \mathcal{C} : \exists (u, v) \in E \text{ s.t. } \chi(u, v) = c\}$. This allows us to check whether

a color is in the palette of an edge $e = (u, v)$ in $O(1)$ time. Specifically, suppose that at the start of round i, we maintain $\overline{P_i(v)}$ for each node v. We can then use this data structure to implement the sampling efficiently as described above. After sampling all of the tentative colors for the edges in S_i, we can then update each of these sets in $O(|S_i|)$ time, by inserting the appropriate colors to each set $\overline{P(v)}$, obtaining $\overline{P_{i+1}(v)}$ for all nodes v. We can also implement data structures that keep track of all the failed edges. We defer all the details of these data structures to Section 10.4.

To summarize, we get the following result.

Lemma 5.4.14. *Given a forest G with maximum degree $\Delta \geq (100 \log n)/\epsilon^4$ as input, the* Nibble *algorithm runs in $O(m/\epsilon^2)$ expected time.*

5.4.3 Analyzing the Nibble Algorithm on General Graphs

In this section, we analyze the Nibble algorithm when the input graph G may contain cycles. Looking back at the analysis in Section 5.4.2, it is easy to check that we crucially needed G to be acyclic because we wanted to apply Observation 5.4.4 in our analysis. This observation was used, for example, in the proof of Lemma 5.4.7 and in Section 5.4.2. The key insight that we employ in this section is that even if G contains cycles, the preceding claim still holds for all the nodes/edges that are *not* part of any cycle of length $\leq T + 1$ (the number of rounds in the Nibble algorithm). Formally, by doing induction on i, we can prove the following lemma.

Lemma 5.4.15. *Let $\mathcal{N}_G(u, j) := G[\{v \in V : dist_G(u, v) \leq j\}]$ denote the j-hop neighborhood of any node $u \in V$. Then for every $i \in [T]$, the palette $P_i(u)$ depends only on: $\mathcal{N}_G(u, i-1)$, and the random bits used by the* Nibble *algorithm to implement rounds $j \in \{1, \ldots, i-1\}$ in $\mathcal{N}_G(u, i-1)$.*[11]

We now introduce a key definition below.

Definition 5.4.16. *We say that a node $v \in V$ is* good *w.r.t. $G = (V, E)$ iff $\mathcal{N}_G(u, T+1)$, the subgraph induced by its $(T+1)$-hop neighborhood in G, does not contain any cycle. Let $U_G \subseteq V$ denote the set of all good nodes in G. We refer to the rest of nodes $B_G := V \setminus U_G$ as* bad *w.r.t. G.*

Informally, since Algorithm 6 only runs for T rounds, and the decisions it makes are *local*, what the algorithm does to some $H \subseteq G$ should only depend on the subgraph of G that is reachable from H by paths of length at most T. Hence, if the $(T+1)$-neighborhood of a node u is acyclic, then it is safe to pretend that the input graph is a forest while analyzing what the algorithm does to the edges incident on u. This implies that even if G contains cycles, we can recover the guarantee of Lemma 5.4.3 for all the good nodes, which leads us to the following lemma.

Lemma 5.4.17. *Suppose that the graph G has maximum degree $\Delta \geq \frac{(100 \log n)}{\epsilon^4}$. Then w.h.p. we have:*

$$deg_F(v) = O(\epsilon\Delta) \text{ for all nodes } v \in U_G.$$

[11] We use the symbol $\text{dist}_G(u, v)$ to denote the distance between u and v in G. Furthermore, for any subset of nodes $V' \subseteq V$, the symbol $G[V']$ denotes the subgraph of G induced by V'.

5.4.4 The Final (Static) Algorithm: NIBBLE on Subsampled Graphs

Throughout Section 5.4.4, we use two parameters: $\gamma := 1/(30T)$ and $\Delta' := \Delta^{\gamma}$. We also assume that:

$$\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}.$$

We now combine Algorithm 6 with the subsampling technique of [KLS+22], which leads to our final algorithm on general graphs. This consists of two steps.

Step I: Set $\eta := \Delta/\Delta'$, and partition the input graph $G = (V, E)$ into η subgraphs $\mathcal{G}_1 = (V, \mathcal{E}_1), \ldots, \mathcal{G}_\eta = (V, \mathcal{E}_\eta)$, by placing each edge $e \in E$ uniformly and independently at random into one of the subsets $\mathcal{E}_1, \ldots, \mathcal{E}_\eta$. To ease notation, for all $j \in [\eta]$ we let $\mathcal{U}_j := U_{\mathcal{G}_j}$ and $\mathcal{B}_j := B_{\mathcal{G}_j}$ denote the sets of good and bad nodes in the subgraph $\mathcal{G}_j$ (see Definition 5.4.16). We now derive two important properties of this subsampling step that will be crucially used in Step II.

Claim 5.4.18. *W.h.p., for every $j \in [\eta]$ we have $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta'$.*

Proof Sketch. Fix an index $j \in [\eta]$. Any edge $e \in E$ appears in $\mathcal{G}_j$ with probability $1/\eta$. Hence, the expected degree of any node $v \in V$ in $\mathcal{G}_j$ is at most $\Delta \cdot (1/\eta) = \Delta'$. Since Δ is sufficiently large, from a straightforward application of Chernoff bound we infer that w.h.p. $\deg_{\mathcal{G}_j}(v) \leq (1+\epsilon) \cdot \Delta'$. The claim now follows from a union bound over all $v \in V$ and $j \in [\eta]$. □

Claim 5.4.19. *Consider the graph $G^\star = (V, E^\star)$, where $E^\star := \bigcup_{j=1}^{\eta} \{(u,v) \in \mathcal{E}_j : \{u,v\} \cap \mathcal{B}_j \neq \emptyset\}$ is the set of all edges that are incident on bad nodes in their corresponding subsampled graphs. Then w.h.p., we have $\Delta(G^\star) = o(\Delta)$.*

We defer the proof of Claim 5.4.19 to Section 5.4.4, and proceed to the next step of our algorithm.

Step II: For each $j \in [\eta]$, taking Claim 5.4.18 into account, we invoke Algorithm 6 on $\mathcal{G}_j$ with a palette $\mathcal{C}_j$ (i.e., the set $\mathcal{C}$ in Line 1 of Algorithm 6) of $(1+\epsilon)^2\Delta'$ colors. The palettes $\mathcal{C}_1, \ldots, \mathcal{C}_j$ are mutually disjoint. For each $j \in [\eta]$, let $\mathcal{F}_j$ be the set of *failed edges* at the end of the corresponding invocation of the NIBBLE algorithm on $\mathcal{G}_j$ (i.e., the set F in Line 13 of Algorithm 6). Let $\mathcal{F} := \bigcup_{j=1}^{\eta} \mathcal{F}_j$ be the collection of these failed edges over all $j \in [\eta]$, and let $\mathcal{G}_{\mathcal{F}} := (V, \mathcal{F})$. We color the edges in $\mathcal{G}_{\mathcal{F}}$ using the folklore algorithm from Section 5.3.1, and an extra set of $O(\Delta(\mathcal{G}_{\mathcal{F}}))$ colors.

It is easy to verify that the above algorithm returns a proper edge coloring of G, and that the number of colors it uses is at most: $O(\Delta(\mathcal{G}_{\mathcal{F}})) + \sum_{j=1}^{\eta} |\mathcal{C}_j| = O(\Delta(\mathcal{G}_{\mathcal{F}})) + (1+\epsilon)^2\Delta' \cdot \eta = O(\Delta(\mathcal{G}_{\mathcal{F}})) + (1+\epsilon)^2\Delta' \cdot (\Delta/\Delta') = O(\Delta(\mathcal{G}_{\mathcal{F}})) + (1+\epsilon)^2\Delta$, w.h.p. We upper bound $\Delta(\mathcal{G}_{\mathcal{F}})$ in Claim 5.4.20 below. This implies the main result in this section, which is summarized in Corollary 5.4.21.

Claim 5.4.20. *We have $\Delta(\mathcal{G}_{\mathcal{F}}) = O(\epsilon\Delta)$ w.h.p.*

Proof. Throughout the proof, fix any node $v \in V$. Partition the set of indices $[\eta]$ into two subsets, $J_g := \{j \in [\eta] : v \in \mathcal{U}_j\}$ and $J_b := \{j \in [\eta] : v \in \mathcal{B}_j\}$, depending on whether or not v is a good node

in the corresponding subsampled graph. For each $j \in J_g$, Claim 5.4.18 and Lemma 5.4.17 imply that $\deg_{\mathcal{F}_j}(v) = O(\epsilon\Delta')$ w.h.p. Thus, summing over all $j \in J_g$, we derive the following bound w.h.p.

$$\sum_{j \in J_g} \deg_{\mathcal{F}_j}(v) = |J_g| \cdot O(\epsilon\Delta') \leq \eta \cdot O(\epsilon\Delta') = O(\epsilon\Delta). \tag{5.11}$$

Next, observe that every edge $(u, v) \in \bigcup_{j \in J_b} \mathcal{F}_j$, by definition, contributes to the set $E^\star$. Thus, from Claim 5.4.19, we infer that w.h.p.

$$\sum_{j \in J_b} \deg_{\mathcal{F}_j}(v) \leq \deg_{G^\star}(v) = o(\Delta). \tag{5.12}$$

From Equation (5.11) and Equation (5.12), we get $\deg_{\mathcal{G}_\mathcal{F}}(v) = O(\epsilon\Delta)$ w.h.p. The claim now follows from a union bound over all $v \in V$. □

Corollary 5.4.21. *W.h.p., the above algorithm returns a $(1 + O(\epsilon))\Delta$-coloring of the input graph G.*

It is easy to verify that the subsampling step can be implemented in $O(m)$ time, because all we need to do is decide for each edge $e \in E$ which subgraph $\mathcal{G}_j$ it should appear in. We now implement the Nibble algorithm in each subsampled graph $\mathcal{G}_j$ using the approach outlined in Section 5.4.2. Putting everything together, this implies a $O_\epsilon(m)$ time algorithm for $(1 + \epsilon)\Delta$-edge coloring in a general graph. See Section 10.4 for details.

Proof (Sketch) of Claim 5.4.19

We will use the following lemma, which is an immediate corollary of Lemma 4.2 in [KLS+22].

Lemma 5.4.22. *Let G' be a subgraph of G obtained by sampling each edge in G independently with probability D/Δ, where $D \geq 2$. Then the probability that the g-neighborhood of a node u contains a cycle in G' is at most $3D^{5g}/\Delta$.*

Consider any node $u \in V$. For each $j \in [\eta]$, define $X_j^u \in \{0, 1\}$ to be the indicator random variable for the event that the $(T+2)$-neighborhood of u in the graph $\mathcal{G}_j$ contains a cycle. Thus, we have $X_j^u = 1$ whenever at least one of the neighbors of u in $\mathcal{G}_j$ is contained in $\mathcal{B}_j$. It follows that:

$$\deg_{G^\star}(u) \leq \sum_{j \in [\eta]} X_j^u \cdot |N(u) \cap \mathcal{E}_j| \leq \sum_{j \in [\eta]} X_j^u \cdot \Delta(\mathcal{G}_j). \tag{5.13}$$

Each edge $e \in E$ is sampled in $\mathcal{G}_j$ independently with probability $1/\eta = \Delta'/\Delta$, and we have that $\Delta' \geq 2$ from the definitions of our parameters. Thus, setting $D = \Delta'$ and $g = T + 2$, Lemma 5.4.22 gives us: $\mathbb{E}\left[X_j^u\right] = \Pr\left[X_u^j = 1\right] \leq 3(\Delta')^{5(T+3)}/\Delta = o(1)$. By linearity of expectation, we get:

$$\mathbb{E}\left[\sum_{j \in [\eta]} X_j^u\right] = \eta \cdot o(1) = o(\Delta/\Delta'). \tag{5.14}$$

With some extra effort, we can show that the random variables $\{X_j^u\}_j$ are negatively associated (see Definition 10.5.5). This allows us to apply a Chernoff bound w.r.t. Equation (5.14), and derive that: $\sum_{j\in[\eta]} X_j^u = o(\Delta/\Delta')$ w.h.p. Next, by Claim 5.4.18, w.h.p. we have: $\Delta(\mathcal{G}_j) = O(\Delta')$ for all $j \in [\eta]$. These last two observations, along with Equation (5.13), imply that $\deg_{G^\star}(u) = o(\Delta)$ w.h.p. The claim now follows from a union bound over all nodes $u \in V$.

5.5 Overview of our Dynamic Nibble Algorithm

In this section, we present an overview of our dynamic algorithm for edge coloring (see Theorem 5.3.2). As in Section 5.4, to convey the main insights behind our approach, the arguments here will be often informal and lacking in low-level details. The complete, formal proofs from this section are deferred to Section 10.2 and Section 10.3.

To simplify the presentation, throughout Section 5.5 we will assume that the input graph $G = (V, E)$ always remains a forest with maximum degree at most Δ, throughout the sequence of updates. We will essentially dynamize the static algorithm from Section 5.4.1. Thus, it is reasonable to expect that once we have a fast dynamic implementation of Algorithm 6 on forests, we can extend this to work on general graphs using the subsampling framework from Section 5.4.4.[12]

Organization. In Section 5.5.1, we present our dynamic algorithm and analyze its *recourse* – the number of changes it makes to the colors of edges per update – ignoring any concern about efficient data structures (see Theorem 5.5.1). Section 5.5.2 shows that a slightly modified version of the dynamic algorithm from Section 5.5.1 can be implemented with fast data structures in $O_\epsilon(1)$ update time.

5.5.1 Bounding the Recourse

We start by presenting our dynamic algorithm.

Preprocessing. We refer to an unordered pair of nodes as a *potential edge*. For each potential edge $e \in \binom{V}{2}$, we independently sample an index $i_e \in [T+1]$ from a *capped geometric distribution* with success probability ϵ, and an (infinite-length) *color-sequence* c_e. Specifically, for each integer $j \geq 1$, the j^{th} color in the sequence c_e is denoted by $c_e(j)$ and is sampled independently and u.a.r. from the palette $\mathcal{C} = [(1+\epsilon)\Delta]$. At preprocessing, we sample and then fix these indices $\{i_e\}_e$ and the color-sequences $\{c_e\}_e$ for every potential edge $e \in \binom{V}{2}$. Note that they uniquely determine the output of Algorithm 6 on any given input graph $G = (V, E)$, as follows. (1) Each edge $e \in E$ selects itself in round i_e (i.e., $e \in S_{i_e}$ if $i_e \leq T$ and $e \in F_{T+1}$ otherwise). (2) While considering an edge $e \in S_i$ in round i we simply scan through the sequence c_e until we find the smallest index $\ell_e \in \mathbb{Z}^+$ such that $c_e(\ell_e) \in P_i(e)$, and set $\tilde{\chi}(e) := c_e(\ell_e)$. If no such index ℓ_e exists, i.e., if $P_i(e) = \emptyset$, then we set $\ell_e := 0$ and $\tilde{\chi}(e) := \perp$.

[12] Note that it is very easy to implement the subsampling based partitioning from Section 5.4.4 in a dynamic setting: When an edge e is inserted, just sample an index $j \in [\eta]$ u.a.r. and add e to the subgraph $\mathcal{G}_j$.

Handling the Sequence of Updates. We use the superscript t to denote the status of any object at time t. For instance, $G^{(t)} = \left(V, E^{(t)}\right)$ denotes the input graph just after the t^{th} update. We maintain the tentative coloring $\tilde{\chi}^{(t)} : E^{(t)} \to \mathcal{C}$, which is obtained by executing Algorithm 6 on $G^{(t)}$, with the same random choices that were fixed at preprocessing. We also collect the subgraph $G_F^{(t)} := \left(V, F^{(t)}\right)$ consisting of all the failed edges and maintain a $O\left(\Delta(G_F^{(t)})\right)$-coloring $\psi^{(t)}$ in $G_F^{(t)}$ using an extra palette of colors that is mutually disjoint with $\mathcal{C}$. The final coloring χ is then defined as follows. For all $e \in E^{(t)}$, we have $\chi^{(t)}(e) = \tilde{\chi}^{(t)}(e)$ if $e \notin F^{(t)}$, and $\chi^{(t)}(e) = \psi^{(t)}(e)$ otherwise.

Theorem 5.5.1. *The dynamic algorithm presented above has an expected recourse of $O(1/\epsilon^4)$ per update, and at each time t w.h.p. maintains a proper $(1 + O(\epsilon))\Delta$-edge coloring of $G^{(t)}$.*

We devote the rest of Section 5.5.1 to the proof of Theorem 5.5.1. Towards this end, first observe that the total number of colors used by the algorithm is $|\mathcal{C}|+O\left(\Delta(G_F^{(t)})\right) = (1+\epsilon)\Delta+O\left(\Delta(G_F^{(t)})\right)$, which equals $(1 + O(\epsilon))\Delta$ w.h.p., according to Lemma 5.4.3. Thus, it now remains to bound the expected recourse of the algorithm. Let $A^{(t)} := \left\{e \in E^{(t-1)} \cup E^{(t)} : \tilde{\chi}^{(t-1)}(e) \neq \tilde{\chi}^{(t)}(e)\right\}$ denote the set of edges that change their tentative color due to the t^{th} update, where we define $\tilde{\chi}^{(t)}(e) = \perp$ for all $e \notin E^{(t)}$. Theorem 5.5.1 then follows from Lemma 5.5.2 and Lemma 5.5.3.

Lemma 5.5.2. *The recourse of the algorithm at time t is at most $O(1) + O\left(\left|A^{(t)}\right|\right)$.*

Proof. W.l.o.g., suppose that the t^{th} update involves the insertion of an edge $e^\star$ (we can analyze deletions analogously). Thus, we have $E^{(t)} := E^{(t-1)} \cup \{e^\star\}$. Note that $\chi^{(t')}(e) \in \{\tilde{\chi}^{(t')}(e), \psi^{(t')}(e)\}$ for all $e \in E^{(t)}$ and $t' \in \{t-1, t\}$.[13] Let $R^{(t)}$ be the recourse of the algorithm for the t^{th} update, and let $R_{\tilde{\chi}}^{(t)}$ and $R_{\psi}^{(t)}$ respectively denote the number of changes to the colorings $\tilde{\chi}$ and ψ due to the t^{th} update. Thus, we have $R^{(t)} \leq R_{\tilde{\chi}}^{(t)} + R_{\psi}^{(t)} = |A^{(t)}| + R_{\psi}^{(t)}$. It now remains to show that $R_{\psi}^{(t)} = O(1) + O\left(|A^{(t)}|\right)$. Towards this end, we first observe that $R_{\psi}^{(t)} \leq |F^{(t-1)} \oplus F^{(t)}|$,[14] because it is trivial to maintain a $O(\Delta)$-coloring in a dynamic graph with maximum degree Δ with a recourse of at most one per update. The lemma now follows from Equation (5.15).

$$|F^{(t-1)} \oplus F^{(t)}| = O(1) + O\left(|A^{(t)}|\right). \tag{5.15}$$

We devote the rest of the proof towards showing why Equation (5.15) holds. Consider any subset $E' \subseteq \binom{V}{2}$ and any coloring $\chi' : E' \to \mathcal{C} \cup \{\perp\}$. We say that an $f \in E'$ is a *failed edge* w.r.t. (E', χ') iff either $\chi'(f) = \perp$ or it has a neighboring edge $e \in E'$ (i.e., e and f shares an endpoint) with the same color (i.e., $\chi'(e) = \chi'(f)$). For instance, $F^{(t)}$ is the set of failed edges w.r.t. $(E^{(t)}, \tilde{\chi}^{(t)})$.

Let $A^{(t)} \cup \{e^\star\} := \{e_1, \ldots, e_\ell\}$, and for each $r \in [0, \ell]$, let $F(r)$ be the set of failed edges w.r.t. $E^{(t)}$ and the coloring where each edge $e \in \{e_1, \ldots, e_r\}$ receives the color $\tilde{\chi}^{(t)}$ and each edge $e \in E^{(t)} \setminus \{e_1, \ldots, e_r\}$ receives the color $\tilde{\chi}^{(t-1)}$. Intuitively, consider a process where we start with the coloring $\tilde{\chi}^{(t-1)}$, and then change the colors of the edges $\{e_1, \ldots, e_\ell\}$ from $\tilde{\chi}^{(t-1)}$ to $\tilde{\chi}^{(t)}$, one at

[13] For each of χ, $\tilde{\chi}$, and ψ, we define the color of an edge e not in the graph as $\perp$.

[14] The symbol $\oplus$ denotes the symmetric difference between two sets.

a time. The sets $F(r)$ track how $F^{(t-1)}$ evolves into $F^{(t)}$ during this process. It is easy to see that

$$|F^{(t-1)} \oplus F^{(t)}| \leq |F(0) \oplus F(1)| + |F(1) \oplus F(2)| + \cdots + |F(\ell-1) \oplus F(\ell)|,$$

since each $f \in F^{(t-1)} \oplus F^{(t)}$ must either be added to or removed from the set of failed edges after some edge $e \in \{e_1, \ldots, e_\ell\}$ changes its color during the process described above. Now, fix any $r \in [\ell]$, let $e_r = (u, v)$, and consider the event when we change the color of e_r from $\tilde{\chi}^{(t-1)}(e_r)$ to $\tilde{\chi}^{(t)}(e_r)$ during the above process (assume for now that neither color is $\perp$). Due to this event, at most 2 edges can get added to the set F. This is because the only edges that will be added to F after this color change are edges incident to u and v with color $\tilde{\chi}^{(t)}(e_r)$ that are not already in F, and there can be at most one such edge incident on each of u and v. By an analogous argument, at most 2 edges can get removed from F due to this event. It follows that $|F(r-1) \oplus F(r)| \leq 4$. A similar argument shows that if $\perp \in \{\tilde{\chi}^{(t-1)}(e_r), \tilde{\chi}^{(t)}(e_r)\}$, then $|F(r-1) \oplus F(r)| \leq 3$. Thus, in both cases, we have $|F(r-1) \oplus F(r)| = O(1)$. Summing over all $r \in [\ell]$, this gives us $|F^{(t-1)} \oplus F^{(t)}| = O(\ell)$. Equation (5.15) now follows because $\ell \leq 1 + |A^{(t)}|$. □

Lemma 5.5.3. *At each time t, we have $\mathbb{E}\left[|A^{(t)}|\right] = O(1/\epsilon^4)$.*

Proof. As in the proof of Lemma 5.5.2, w.l.o.g. suppose that the t^{th} update involves the insertion of an edge $e^\star$ (we can analyze deletions analogously). Thus, we have $E^{(t)} := E^{(t-1)} \cup \{e^\star\}$. Let $A_i^{(t)} := A^{(t)} \cap S_i^{(t)}$ for all $i \in [T]$. For the rest of the proof, fix the rounds $\{i_e\}_e$ of all potential edges $e \in \binom{V}{2}$, and condition on the high-probability event $\mathcal{Z}$ (see Lemma 5.4.5) on both $G^{(t-1)}$ and $G^{(t)}$.

Since the edge $e^\star$ gets selected in round $i_{e^\star}$, due to the t^{th} update no edge $e \in S^{(t)}_{\leq i_{e^\star}} \setminus \{e^\star\}$ changes its tentative color. In other words, we have $|A_i^{(t)}| = 0$ for all $i < i_{e^\star}$ and $|A^{(t)}_{i^{e^\star}}| \leq 1$. We will show:

$$\mathbb{E}\left[|A_i^{(t)}|\right] \leq 4\epsilon \cdot \mathbb{E}\left[|A^{(t)}_{<i}|\right] \text{ for all } i \in [i_{e^\star}+1, T]. \tag{5.16}$$

Equation (5.16) implies that $\mathbb{E}[|A^{(t)}|] \leq (1+4\epsilon)^T \leq e^{4\epsilon T} = O(1/\epsilon^4)$. Thus, it now remains to prove Equation (5.16). Towards this end, consider the following scenario where: (i) we fixed the rounds $\{i_e\}_e$ and color-sequences $\{c_e\}_e$ of all potential edges at preprocessing, (ii) we have already computed the outcome of Algorithm 6 on $G^{(t-1)}$ in accordance with these choices $\{i_e, c_e\}_e$, and (iii) now we are running Algorithm 6 again on $G^{(t)}$ with the same choices $\{i_e, c_e\}_e$ and observing which edges get added to $A^{(t)}$ as Algorithm 6 implements the rounds $i = 1, \ldots, T$ on $G^{(t)}$, in this order.

It is easy to observe that no edge gets added to $A^{(t)}$ during rounds $1, \ldots, i_{e^\star}-1$. At round $i = i_{e^\star}$, the edge $e^\star$ is now present and it receives a tentative color (as long as its palette is non-empty). Now, consider any subsequent round $i \in \{i_{e^\star}+1, \ldots, T\}$. At the start of round i, we have already determined the palette $P_i^{(t)}(v)$ for each node $v \in V$. During round i, an edge $e = (u, v) \in S_i^{(t)}$ might get added to $A^{(t)}$, but this can happen only due to one of the following two reasons.

- (I) $c_e(\ell_e^{(t-1)}) \notin P_i^{(t)}(u) \cap P_i^{(t)}(v)$. So, there is an edge $e' \in N^{(t)}_{<i}(e)$ with $\tilde{\chi}^{(t)}(e') = c_e(\ell_e^{(t-1)}) = \tilde{\chi}^{(t-1)}(e)$. Since $\tilde{\chi}^{(t-1)}$ was the output of Algorithm 6 on $G^{(t-1)}$, we have $e' \in A^{(t)}$. In this case, we say that e' is *type-I-responsible* for e being added to $A^{(t)}$.

- (II) There is an $\ell \in \left[\ell_e^{(t-1)} - 1\right]$ such that $c_e(\ell) \in P_i^{(t)}(u) \cap P_i^{(t)}(v)$. Let ℓ' be the smallest such ℓ. Note that Algorithm 6 will set $\tilde{\chi}^{(t)}(e) := c_e(\ell')$ and $\ell_e^{(t)} := \ell'$. As $\tilde{\chi}^{(t-1)}$ was the output of Algorithm 6 on $G^{(t)}$, there is an edge $e' \in N_{<i}^{(t-1)}(e)$ with $\tilde{\chi}^{(t-1)}(e') = c_e(\ell') = \tilde{\chi}^{(t)}(e)$. But since $c_e(\ell') \in P_i^{(t)}(u) \cap P_i^{(t)}(v)$, we now have $\tilde{\chi}^{(t)}(e') \neq c_e(\ell')$, which implies that $e' \in A^{(t)}$. In this case, we say that e' is *type-II-responsible* for e being added to $A^{(t)}$.

Motivated by the preceding two observations, we now define the following sets for each edge $e' \in E^{(t)}$:

$$\Gamma_i(e') := \left\{ e \in N_i^{(t)}(e') \,\middle|\, \tilde{\chi}^{(t)}(e') = \tilde{\chi}^{(t-1)}(e) \right\}, \; \Lambda_i(e') := \left\{ e \in N_i^{(t)}(e') \,\middle|\, \tilde{\chi}^{(t-1)}(e') = \tilde{\chi}^{(t)}(e) \right\}.$$

To summarize, for every edge e that gets added to $A_i^{(t)}$, there is some edge $e' \in A_{<i}^{(t)}$ that is either type-I or type-II responsible for e. Furthermore, if e' is type-I (resp. type-II) responsible for e, then we must have $e \in \Gamma_i(e')$ (resp. $e \in \Lambda_i(e')$). This leads us to the following observation.

$$|A_i^{(t)}| \;\leq \sum_{e' \in A_{<i}^{(t)}} \left(|\Gamma_i(e')| + |\Lambda_i(e')| \right) \text{ for all } i \in [i_{e^\star} + 1, T]. \tag{5.17}$$

We now make the following claim, whose proof is deferred to Section 5.5.1.

Claim 5.5.4. *Consider any round $i \in [i_{e^\star} + 1, T]$ and any edge $e \in S_{<i}^{(t)}$. Then we have:*

$$\mathbb{E}\left[|\Gamma_i(e)| + |\Lambda_i(e)| \,\middle|\, e \in A_{<i}^{(t)} \right] \leq 4\epsilon.$$

For the rest of the proof, fix any round $i \in [i_{e^\star} + 1, T]$. Observe that:

$$\begin{aligned} \mathbb{E}\left[|A_i^{(t)}| \right] &\leq \sum_{e \in S_{<i}^{(t)}} \mathbb{E}\left[|\Gamma_i(e)| + |\Lambda_i(e)| \,\middle|\, e \in A_{<i}^{(t)} \right] \cdot \Pr\left[e \in A_{<i}^{(t)} \right] \\ &\leq 4\epsilon \cdot \sum_{e \in S_{<i}^{(t)}} \Pr\left[e \in A_{<i}^{(t)} \right] = 4\epsilon \cdot \mathbb{E}\left[|A_{<i}^{(t)}| \right]. \end{aligned}$$

In the above derivation, the first inequality follows from Equation (5.17), whereas the second inequality follows from Claim 5.5.4. Thus, Equation (5.16) holds, and this concludes the proof of the lemma. □

Proof of Claim 5.5.4

The occurrence of the event $\left\{ e \in A_{<i}^{(t)} \right\}$ depends only on the color-sequences of those edges $e' \in S_{<i}^{(t)}$ that are in the same connected component as e in $G_{<i}^{(t)} := (V, S_{<i}^{(t)})$. For the rest of the proof, we fix these relevant color-sequences in such a way that the event $\left\{ e \in A_{<i}^{(t)} \right\}$ occurs. Below, we will show that $\mathbb{E}\left[|\Gamma_i(e)| \right] \leq 2\epsilon$. The proof for $\mathbb{E}\left[|\Lambda_i(e)| \right] \leq 2\epsilon$ is completely analogous, and taken together, they imply Claim 5.5.4.

By linearity of expectation, we have:

$$\mathbb{E}\left[|\Gamma_i^{(t)}(e)|\right] = \sum_{f \in N_i^{(t)}(e)} \Pr\left[\tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e)\right]. \tag{5.18}$$

Let $e = (u, v)$. The palettes $P_i^{(t-1)}(u)$ and $P_i^{(t-1)}(v)$ are completely determined by the color-squences we have fixed, because $G_{<i}^{(t-1)}$ is a subgraph of $G_{<i}^{(t)}$ as we are considering the scenario where the t^{th} update is an edge-insertion. Consider an edge $f = (u, w) \in N_i^{(t-1)}(e)$ (note that $N_i^{(t)}(e) = N_i^{(t-1)}(e)$ since $i_{e^\star} < i$). Since the graph $G_i^{(t)}$ contains no cycles, e and w lie in different connected components of $G_{<i}^{(t)}$. Thus, the color $\tilde{\chi}^{(t)}(e)$ is independent of the palette $P_i^{(t-1)}(w)$. Let $c = \tilde{\chi}^{(t)}(e)$. We can now apply Lemma 5.4.2 and Corollary 5.4.6 to get:

$$\Pr\left[\tilde{\chi}^{(t-1)}(f) = c\right] \leq \frac{1}{|P_i^{(t-1)}(u)|} \leq \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}. \tag{5.19}$$

From Equation (5.18) and Equation (5.19), we now derive that:

$$\begin{aligned}
\mathbb{E}\left[|\Gamma_i^{(t)}(e)|\right] &= \sum_{f \in N_i^{(t)}(e)} \Pr\left[\tilde{\chi}^{(t-1)}(f) = c\right] \\
&= \sum_{f \in N_i^{(t-1)}(u)} \Pr\left[\tilde{\chi}^{(t-1)}(f) = c\right] + \sum_{f \in N_i^{(t-1)}(v)} \Pr\left[\tilde{\chi}^{(t-1)}(f) = c\right] \\
&\leq \frac{|N_i^{(t-1)}(u)| + |N_i^{(t-1)}(v)|}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta} \leq \frac{2\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta} = 2\epsilon.
\end{aligned}$$

The penultimate step in the above derivation follows from Lemma 5.4.5.

5.5.2 Bounding the Update Time

We now outline how to implement a modified version of the algorithm in Section 5.5.1, which leads to dynamic $(1+\epsilon)\Delta$-edge coloring in $O(\text{poly}(1/\epsilon))$ update time (see Theorem 5.3.2).

A "Template" Algorithm. We start with a *template algorithm*, which differs from the one in Section 5.5.1 in only one aspect: The color-sequence of every potential edge is now of length $K := \lceil (8/\epsilon^2)\log(1/\epsilon) \rceil$. Specifically, at preprocessing each potential edge e constructs the color-sequence c_e by sampling K, as opposed to infinitely many, colors $c_e(1), \ldots, c_e(K)$ from the palette $\mathcal{C} = [(1+\epsilon)\Delta]$, independently and u.a.r. Subsequently, the algorithm handles the graph $G^{(t)}$ at time t as follows. While executing round $i \in [T]$ of the NIBBLE algorithm on $G^{(t)}$, an edge $e \in S_i^{(t)}$ picks the minimum index $\ell \in [K]$ s.t. $c_e(\ell) \in P_i^{(t)}(u) \cap P_i^{(t)}(v)$, and sets $\tilde{\chi}^{(t)}(e) := c_e(\ell)$ and $\ell_e^{(t)} := \ell$. If there is no such ℓ, then it sets $\tilde{\chi}^{(t)}(e) := \perp$ and $\ell_e^{(t)} := 0$. Everything else remains the same as in Section 5.5.1. We now give an intuitive justification as to why the above modification does not (meaningfully) change the guarantees derived in Section 5.5.1 (see Section 10.1 for a formal argument).

An Intuitive Justification. Fix the rounds $\{i_e\}_e$ of all potential edges $e \in \binom{V}{2}$, and condition on the high-probability event $\mathcal{Z}$ (see Lemma 5.4.5) on the current graph $G^{(t)}$. Consider any round $i \in [T]$ and any edge $e \in S_i^{(t)}$. By Corollary 5.4.8, we have that $|P_i(e)| \geq \epsilon^2(1+\epsilon)\Delta/8$ w.h.p. As $|\mathcal{C}| = (1+\epsilon)\Delta$, the probability that the color-sequence c_e does not contain some color from $P_i(e)$ is given by: $\left(1 - \frac{\epsilon^2(1+\epsilon)\Delta}{8\cdot|\mathcal{C}|}\right)^K \leq \left(1 - \frac{\epsilon^2}{8}\right)^K \leq \left(1 - \frac{\epsilon^2}{8}\right)^{(8/\epsilon^2)\log(1/\epsilon)} \leq \epsilon$. To summarize, with probability at most ϵ, the template algorithm mistakenly sets $\tilde{\chi}^{(t)}(e) = \perp$. But with probability $1-\epsilon$, it correctly sets $\tilde{\chi}^{(t)}(e)$ to be a color chosen u.a.r. from $P_i^{(t)}(e)$. Thus, the template algorithm is almost similar to the original algorithm, except that each edge $e \in E^{(t)}$, before even being considered for receiving a tentative color, gets added to the failed set $F^{(t)}$ with probability ϵ. This increases the maximum degree of the subgraph $G_F^{(t)} := (V, F^{(t)})$ by at most $\epsilon\Delta$ (we can show that this bound holds w.h.p.). Since anyway we maintain a $O\left(G_F^{(t)}\right)$-coloring in $G_F^{(t)}$ using a separate palette, the total number of colors needed by the algorithm continues to remain $(1 + O(\epsilon))\Delta$.

Data Structures. In order to highlight the main ideas behind our data structures, we start by making two simplifications. (I) We allow for a preprocessing time of $O_\epsilon(n^2)$. This means that we can afford to fix the rounds $\{i_e\}_e$ and the color-sequences $\{c_e\}_e$ for every potential edge $e \in \binom{V}{2}$ at preprocessing. (II) We focus only on maintaining the tentative coloring $\tilde{\chi}^{(t)}$ on $G^{(t)}$. This allows us to ignore keeping track of the set of failed edges $F^{(t)}$, and the coloring $\psi^{(t)}$ on $G_F^{(t)} := (V, F^{(t)})$.

Towards the end of this section, we explain how we can easily get rid of the preprocessing time, provided we start with an empty graph $G^{(0)} = (V, E^{(0)})$. We defer presenting the data structures which maintain the coloring $\psi^{(t)}$ to the full version in Section 10.3.

We are now ready to define the key data structure. For all nodes $v \in V$, rounds $i \in [T]$ and colors $c \in \mathcal{C}$, let $\mathcal{L}_{v,i}^{(t)}(c) := \left\{ e \in N_i^{(t)}(v) \;\middle|\; \exists k \in [K] \text{ s.t. } c_e(\ell_k) = c \right\}$ denote the set of all neighboring edges $e \in N_i^{(t)}(v)$ such that the color c appears at least once in the color-sequence c_e. We store all these sets $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$ as hash-tables. In the two claims below, we bound the sizes of these sets.

Claim 5.5.5. *For each $v \in V, i \in [T]$ and $c \in \mathcal{C}$, we have $\mathbb{E}\left[|\mathcal{L}_{v,i}^{(t)}(c)|\right] = K = O_\epsilon(1)$.*

Proof Sketch. Consider any edge $e \in N_i^{(t)}(v)$. The expected number of times the color c appears in the sequence c_e is given by $K/|\mathcal{C}| = K/((1+\epsilon)\Delta) \leq K/\Delta$. Hence, by Markov's inequality, we have $\Pr\left[e \in \mathcal{L}_{v,i}^{(t)}(c)\right] \leq K/\Delta$. Since $|N_i^{(t)}(v)| \leq \Delta$, it follows that $\mathbb{E}\left[|\mathcal{L}_{v,i}^{(t)}(c)|\right] \leq (K/\Delta)\cdot\Delta = K$. □

Claim 5.5.6. *We always have $\sum_{v\in V, i\in[T], c\in\mathcal{C}} |\mathcal{L}_{v,i}^{(t)}(c)| = O\left(KT\cdot|E^{(t)}|\right) = O_\epsilon\left(|E^{(t)}|\right)$. Further, insertion/deletion of an edge e in G can lead to at most $2KT = O_\epsilon(1)$ changes in the sets $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$.*

Proof Sketch. Each edge $e = (u, w) \in E^{(t)}$ can appear in at most $2KT$ sets $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$, one for each of its endpoints $x \in \{u, w\}$, for each round $i \in [T]$, and most importantly, for each of the (at most) K colors that appear in its color-sequence c_e. For the same reason, it is also the case that the insertion/deletion of an edge (u, v) in G can affect at most $2KT$ of the sets $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$. □

Claim 5.5.6 implies that we can maintain the hash-tables for $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$ in $O_\epsilon(1)$ update time (note that these hash-table don't depend at all on the tentative coloring $\tilde{\chi}^{(t)}$), and that the total space complexity of this data structure is $O_\epsilon\left(|E^{(t)}|\right)$. We now show how using this data structure we can implement the template algorithm in expected update time proportional to its recourse, which in turn, is $O(1/\epsilon^4)$ according to Theorem 5.5.1.

Modifying the Coloring $\tilde{\chi}$ After the t^{th} Update. Recall the notations from Section 5.5.1, and consider the following scenario. We already have the coloring $\tilde{\chi}^{(t-1)}$ on $G^{(t-1)}$, when we receive the t^{th} update to G. Our goal now is to change the coloring $\tilde{\chi}^{(t-1)}$ into $\tilde{\chi}^{(t)}$.[15] To achieve this goal, all we need to do is consider the rounds $i = 1, \ldots, T$, one at a time. And while we are at round $i \in [T]$, we need to identify the set $A_i^{(t)}$ and change the color of each edge $e \in A_i^{(t)}$ appropriately.

We claim that once we have identified an edge $e \in A_i^{(t)}$ during round i, changing its color takes only $O(K) = O_\epsilon(1)$ time. To see why this is true, note that we can easily maintain the complements of the palettes $\left\{\overline{P_i(v)}\right\}_v$ for all nodes $v \in V$ as hash-tables, where $\overline{P_i(v)} := \mathcal{C} \setminus P_i(v)$.[16] Now, for the edge $e = (u, v) \in A_i^{(t)}$, we scan through its color-sequence c_e and identify the smallest index (if any) $\ell \in [K]$ such that $c_e(\ell) \notin \overline{P_i^{(t)}(u)} \cup \overline{P_i^{(t)}(v)}$. This takes $O_\epsilon(1)$ time per index in $[K]$, because at the start of round i we already have access to the hash tables for $\overline{P_i^{(t)}(u)}$ and $\overline{P_i^{(t)}(v)}$.

Thus, the time spent on implementing round i is dominated by the time spent on identifying the set $A_i^{(t)}$ in the first place. We will now show that using our data structures we can perform this task in expected time $O_\epsilon\left(|A_{<i}^{(t)}|\right)$. Summing over $i \in [T]$, this will lead to an expected update time of $\sum_{i\in[T]} O_\epsilon\left(|A_{<i}^{(t)}|\right) = O_\epsilon\left(\sum_{i\in[T]} T \cdot |A_i^{(t)}|\right) = O_\epsilon(|A^{(t)}|)$, which is $O_\epsilon(1)$ in expectation, according to Lemma 5.5.3. It now only remains to prove the claim below.

Claim 5.5.7. *While implementing round i of the template algorithm after the t^{th} update, we can identify the set $A_i^{(t)}$ in expected time $O_\epsilon(|A_{<i}^{(t)}|)$.*

Proof Sketch. From the proof of Lemma 5.5.3 (specifically, from the discussion in the paragraph preceding Equation (5.17)), it follows that an edge $e \in S_i^{(t)}$ belongs to $A_i^{(t)}$ only if it has a neighbor $e' \in N_{<i}^{(t)}(e) \cap A_{<i}^{(t)}$ such that either $e \in \Gamma_i^{(t)}(e')$ or $e \in \Lambda_i^{(t)}(e')$. Note that in the former (resp. latter) case, the color $\tilde{\chi}^{(t)}(e')$ (resp. $\tilde{\chi}^{(t-1)}(e')$) must appear at least once in the color-sequence c_e. In other words, an edge $e = (u, v) \in S_i^{(t)}$ appears in $A_i^{(t)}$ only if it has a neighboring edge $e' = (v, w) \in N_{<i}^{(t)}(e) \cap A_{<i}^{(t)}$ such that $e \in \mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t)}(e')\right) \cup \mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t-1)}(e')\right)$. This motivates the following approach for identifying the set $A_i^{(t)}$ at the start of round i. Construct a set $\mathcal{A}_i^{(t)}$ by taking the union, over all $e' = (v, w) \in A_{<i}^{(t)}$, of the sets $\mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t)}(e')\right) \cup \mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t-1)}(e')\right)$. By Claim 5.5.5, this takes $O_\epsilon\left(|A_{<i}^{(t)}|\right)$ expected time. We also know for sure that $A_i^{(t)} \subseteq \mathcal{A}_i^{(t)}$, and $\mathbb{E}\left[|\mathcal{A}_i^{(t)}|\right] = O_\epsilon\left(|A_{<i}^{(t)}|\right)$.

We now scan through all the edges $e \in \mathcal{A}_i^{(t)}$, and for each of them determine whether or not it belongs to $A_i^{(t)}$. This takes $O(K) = O_\epsilon(1)$ expected time per edge in $\mathcal{A}_i^{(t)}$, because given an

[15] Note that $\tilde{\chi}^{(t-1)}$ (resp. $\tilde{\chi}^{(t)}$) is the output of the template algorithm on $G^{(t-1)}$ (resp. $G^{(t)}$).

[16] For each color $c \in \overline{P_i(v)}$, maintain a counter which denotes the number of edges in $N_{<i}(v)$ that receive c as their tentative color, and update these counters accordingly whenever an edge changes its tentative color.

edge $e = (u, v) \in \mathcal{A}_i^{(t)}$ all we need to do is to check which of the colors $\{c_e(\ell)\}_{\ell \in [K]}$ belong to $P_i^{(t)}(u) \cap P_i^{(t)}(v) = \mathcal{C} \setminus \left(\overline{P_i^{(t)}(u)} \cup \overline{P_i^{(t)}(v)} \right)$, and this takes $O(1)$ expected time per color.

To summarize, we can indeed identify the set $A_i^{(t)}$ in $O_\epsilon\left(|A_{<i}^{(t)}|\right)$ expected time. □

Getting Rid of the Preprocessing Time. Suppose that we start with an empty graph $G^{(0)} := (V, E^{(0)})$ before any update arrives. We can get rid of the $O_\epsilon(n^2)$ preprocessing by making the following simple observation: We don't need to fix the rounds and color-sequences of potential edges in advance. We can do that *on the fly*. Specifically, when an edge e gets inserted, we sample its round i_e and color-sequence c_e in $O_\epsilon(1)$ time. We work with (i_e, c_e) as long as the edge e is present in the input graph. When the edge e gets deleted, we remove all information/storage about (i_e, c_e). If the edge e gets inserted again at some point in future, then we sample a fresh pair (i_e, c_e) and work with this new sample. It is easy to verify that all our recourse and update time bounds, as well as the guarantee on the total number of colors used, continue to hold if we make this simple modification. The only effect it has is that now we don't need to worry about preprocessing time at all, provided we are starting with an empty graph.

Chapter 6

Open Questions and Future Directions

In this chapter, we discuss open problems and future research directions to build on the work presented in this thesis.

Static Edge Coloring

Our near-linear $O(m \log \Delta)$ time algorithm for computing a $(\Delta+1)$-coloring in Theorem 1.1.1 settles the complexity of this problem up to a $O(\log \Delta)$ factor. A natural question is whether or not we can push this all the way down to $O(m)$ and obtain an optimal linear time algorithm.

Open Problem 1. *Can we design a $O(m)$ time algorithm for $(\Delta+1)$-coloring?*

Even the classic bipartite Δ-coloring algorithm of [COS01] has a running time of $O(m \log \Delta)$, and getting a running time of $O(m)$ for bipartite graphs is still wide open. At a high level, the main challenge in shaving this final $O(\log \Delta)$ factor is that many of the techniques used to obtain near-linear time algorithms inherently lead to a polylogarithmic term, such as Euler partitioning, which has a recursion depth of $O(\log \Delta)$, or color extension subroutines that can handle at most a 'constant fraction' of edges at a time (Lemma 7.2.3), and thus must be applied $\Omega(\log n)$ times. Even though we were able to consolidate these separate polylogarithmic factors into a single $O(\log \Delta)$ term (see Section 7.6), obtaining a running time of $O(m)$ would require overcoming all of these obstacles. Therefore, getting a linear time algorithm might require taking a very different approach.

Parallel Edge Coloring

The shift towards processing massive datasets has necessitated the development of algorithms in models that capture the challenges of large-scale computation, such as parallelism and communication bottlenecks [KSV10]. In these settings, the goal is not primarily to minimize the total work, but to minimize the parallel time or rounds of communication. A major focus in parallel computing is designing algorithms with low *depth*, which corresponds to fast running times in the *parallel random-access machine* (PRAM) model. In this setting, we have polynomially many processors with

access to a shared memory, and the objective is to minimize the number of rounds of computation, while ideally also minimizing the total work.

The first parallel algorithms for edge coloring were designed by [KS87]. Their work presented multiple simple algorithms, including a deterministic algorithm that computes a $(\Delta+1)$-coloring in $\text{poly}(\Delta, \log n)$ depth with $\tilde{O}(n \cdot \text{poly}(\Delta))$ total work, as well as a randomized algorithm that computes a $(\Delta + \tilde{O}(\sqrt{\Delta}))$-coloring in $\text{polylog}(n)$ depth with $\tilde{O}(m)$ total work. Subsequent improvements on their parallel $(\Delta+1)$-coloring algorithm gave improvements to the bound on the depth [LSH96], but failed to remove the polynomial dependence on Δ.

Understanding the parallel complexity of $(\Delta+1)$-coloring, and whether or not we can compute such a coloring in $\text{polylog}(n)$ depth, is a significant open problem. In light of our recent progress in the sequential setting, a natural extension is to design a parallel algorithm with polylogarithmic depth that also has near-linear total work, effectively creating a parallel counterpart to the near-optimal static algorithm given in Theorem 1.1.1. This leads us to the following question.

Open Problem 2. *Can we design a* $\text{polylog}(n)$ *depth parallel algorithm for computing a* $(\Delta+1)$*-coloring with* $\tilde{O}(m)$ *total work?*

Dynamic Edge Coloring

In the dynamic setting, we are still far from settling the complexity of edge coloring; for $(\Delta+1)$-coloring, the state-of-the-art update time is $\text{poly}(\Delta, \log n)$, while in the static setting, we can achieve a running time that is near-linear. Obtaining an update time of $\text{polylog}(n)$ for maintaining a $(\Delta+1)$-coloring would close this gap. While we are still far from obtaining such a result, a natural starting point is to understand whether we can efficiently obtain an *additive* approximation instead of a *multiplicative* approximation. In particular, we have the following open problem.

Open Problem 3. *Can we maintain a dynamic* $(\Delta + \tilde{O}(\Delta^{1-\gamma}))$*-coloring with* $\text{polylog}(n)$ *update time, for some constant* $\gamma > 0$*?*

This question is open even if we only consider the recourse of the coloring instead of the update time. Due to the lower bound of [CHL+20] (see Theorem 5.1.1), many of the standard techniques used to design edge coloring algorithms will not be enough to give an algorithm that can answer this question [DHZ19, Chr23, Chr26]. Furthermore, it may be possible to obtain a recourse lower bound in the fully dynamic setting and rule out the possibility of obtaining such a result.

Another related question is whether we can design an algorithm that extends a $(\Delta+1)$-coloring to an uncolored edge in $\tilde{O}(\Delta)$ time, matching the lower bound of [CHL+20] for $(\Delta+1)$-coloring.

Open Problem 4. *Can we extend a* $(\Delta+1)$*-coloring to an uncolored edge in* $\tilde{O}(\Delta)$ *time?*

While we made progress towards answering this question with Theorem 1.1.6, the core argument used to design multi-step Vizing chains is not capable of giving chains with length $o(\Delta^2)$, and thus can only be used to obtain algorithms that run in $\Omega(\Delta^2)$ time. Obtaining such a result would likely require some significant new insights, making this an interesting research direction. Additionally,

this result would immediately lead to a different near-linear time algorithm for $(\Delta+1)$-coloring (see Section 2.2.1).

Part II

Fast Static Edge Coloring

Chapter 7

Vizing's Theorem in Randomized Near-Linear Time

In this chapter, we give our full randomized near-linear time algorithm for $(\Delta+1)$-coloring, proving Theorem 1.1.1, which we restate below.

Theorem 1.1.1. *There is an algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $O(m \log \Delta)$ time with high probability.*

Our main contribution in this work is to improve the time-complexity of $(\Delta+1)$-edge coloring by polynomial factors all the way to near-linear. For this reason, as well as for the sake of transparency of our techniques, we focus primarily on presenting an $O(m \log^3 n)$ time randomized algorithm (Theorem 7.2.1), which showcases our most novel ideas. Later, in Section 7.6, we show that a more careful implementation of the same algorithm achieves a clean $O(m \log n)$ runtime, giving the following result.

Theorem 7.0.1. *There is an algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $O(m \log n)$ time with high probability.*

In Section 7.8, we show how to replace the $\log n$ term in Theorem 7.0.1 with a $\log \Delta$ term, leading to an algorithm for $(\Delta+1)$-coloring in $O(m \log \Delta)$ time, proving Theorem 1.1.1.

In Sections 7.7 and 7.8, we show how to extend our result to Vizing's theorem for multigraphs, proving Theorem 1.1.3, which we restate below.

Theorem 1.1.3. *There is an algorithm that, given a multigraph G with maximum degree Δ and maximum multiplicity μ, computes a $(\Delta+\mu)$-coloring of G in $O(m \log \Delta)$ time with high probability.*

Notation. This chapter uses the notation described in Chapter 3. We introduce the rest of the notation used throughout this chapter in Section 7.1.

7.1 Basic Building Blocks

In this section, we introduce the notion of u-fans, u-edges, and separable collections, which are the definitions that work as the basic building blocks for our algorithms.

7.1.1 U-Fans and U-Edges

We begin by defining the notion of *u-fans* that was used by [GNK+85].[1]

Definition 7.1.1 (u-fan, [GNK+85])**.** *A* u-fan *is a tuple* $\boldsymbol{f} = (u, v, w, \alpha, \beta)$ *where* u*,* v *and* w *are distinct vertices and* α *and* β *are distinct colors such that:*

1. (u, v) *and* (u, w) *are uncolored edges.*
2. $\alpha \in \mathsf{miss}_\chi(u)$ *and* $\beta \in \mathsf{miss}_\chi(v) \cap \mathsf{miss}_\chi(w)$*.*

We say that u is the *center* of $\boldsymbol{f}$ and that v and w are the *leaves* of $\boldsymbol{f}$. We also say that the u-fan $\boldsymbol{f}$ is $\{\alpha, \beta\}$-*primed* and define $c_{\boldsymbol{f}}(u) := \alpha$, $c_{\boldsymbol{f}}(v) := \beta$ and $c_{\boldsymbol{f}}(w) := \beta$ (i.e. given a vertex $x \in \boldsymbol{f}$, $c_{\boldsymbol{f}}(x)$ is the available color that $\boldsymbol{f}$ 'assigns' to x).

Activating U-Fans. Let $\boldsymbol{f}$ be an $\{\alpha, \beta\}$-primed u-fan with center u and leaves v and w. The key property of u-fans is that at most one of the $\{\alpha, \beta\}$-alternating paths starting at v or w ends at u. Say that the $\{\alpha, \beta\}$-alternating path starting at v does not end at u. Then, after flipping this $\{\alpha, \beta\}$-alternating path, both u, v are missing color α. Thus, we can extend the coloring χ by assigning $\chi(u, v) \leftarrow \alpha$. We refer to this as *activating* the u-fan $\boldsymbol{f}$.

We also define the notion of a *u-edge* similarly to u-fans.

Definition 7.1.2 (u-edge)**.** *A* u-edge *is a tuple* $\boldsymbol{e} = (u, v, \alpha)$ *where* (u, v) *is an uncolored edge and* α *is a color such that* $\alpha \in \mathsf{miss}_\chi(u)$*.*

We say that u is the *center* of $\boldsymbol{e}$ and that α is the *active color* of $\boldsymbol{e}$. For notational convenience, we also say that the u-edge $\boldsymbol{e}$ is α-*primed* and define $c_{\boldsymbol{e}}(u) := \alpha$ and $c_{\boldsymbol{e}}(v) = \perp$.[2]

Collections of U-Components. While working with both u-fans and u-edges simultaneously, we sometimes refer to some $\boldsymbol{g}$ that is either a u-fan or a u-edge as a *u-component*. Throughout this chapter, we often consider collections of u-components $\mathcal{U}$. We only use the term 'collection' in this context, so we abbreviate 'collection of u-components' by just 'collection'. We will be particularly interested in collections satisfying the following useful property, which we refer to as *separability*.

Definition 7.1.3 (Separable Collection)**.** *Let* χ *be a partial* $(\Delta + 1)$*-edge coloring of* G *and* $\mathcal{U}$ *be a collection of u-components (i.e. u-fans and u-edges). We say that the collection* $\mathcal{U}$ *is* separable *if the following holds:*

[1] The term 'u-fan' was originally introduced by [GNK+85] as an abbreviation for 'uncolored fan'.

[2] Whenever we refer to an "uncolored edge e", we are referring to an edge $e \in E$ such that $\chi(e) = \perp$, whereas a 'u-edge $\boldsymbol{e}$' always refers to the object from Definition 7.1.2 and is denoted in bold.

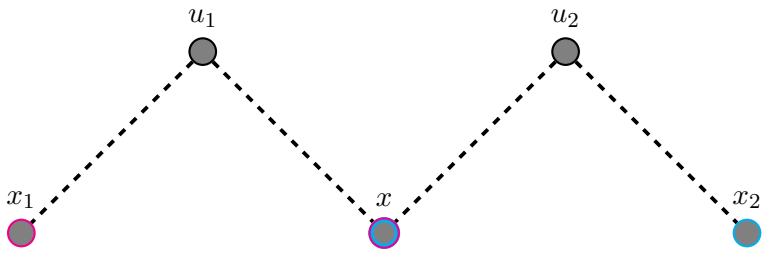

Figure 7.1: This picture shows two u-fans $(u_1, x_1, x, \cdot, \beta_1)$ and $(u_2, x_2, x, \cdot, \beta_2)$ sharing a common vertex x. The separable condition requires that $\beta_1 \neq \beta_2$; for instance β_1, β_2 could be magenta and cyan as shown here.

1. *All of the u-components in $\mathcal{U}$ are edge-disjoint.*

2. *For each $x \in V$, the colors in the multi-set $C_{\mathcal{U}}(x) := \{c_{\boldsymbol{g}}(x) \mid \boldsymbol{g} \in \mathcal{U}, x \in \boldsymbol{g}\}$ are distinct.*

We remark that the second property of this definition is rather important because we need to ensure that different u-components are not interfering with each other when they share common vertices. Check Figure 7.1 for an illustration.

Damaged U-Components. Suppose we have a partial $(\Delta+1)$-edge coloring χ and a separable collection $\mathcal{U}$ w.r.t. χ. Now, suppose that we modify the coloring χ. We say that a u-component $\boldsymbol{g} \in \mathcal{U}$ is *damaged* if it is no longer a u-component w.r.t. the new coloring χ.

We note that this can only happen for one of the following two reasons: (1) one of the uncolored edges in $\boldsymbol{g}$ is now colored, or (2) there is a vertex $x \in \boldsymbol{g}$ such the color $c_{\boldsymbol{g}}(x)$ that $\boldsymbol{g}$ assigns to x is no longer missing at x.

The following lemma shows that flipping the colors of an alternating path cannot damage many u-components in a separable collection $\mathcal{U}$.

Lemma 7.1.4. *Let χ be a partial $(\Delta+1)$-edge coloring of a graph G, $\mathcal{U}$ a separable collection and P a maximal alternating path in χ. Then flipping the colors of the alternating path P damages at most 2 u-components in $\mathcal{U}$ (corresponding to the two endpoints of the path).*

Proof. Let x and y be the endpoints of the path P. Since only the palettes of x and y change after flipping the colors of P, only u-components $\boldsymbol{g} \in \mathcal{U}$ that contain x and y may be damaged. Since the palette of x changes by 1 and the collection $\mathcal{U}$ is separable, only one $\boldsymbol{g} \in \mathcal{U}$ containing x may be damaged by the color $c_{\boldsymbol{g}}(x)$ no longer being available at x. The same holds for the vertex y. Thus, at most 2 u-components in $\mathcal{U}$ are damaged by this operation. □

7.1.2 Data Structures

In Section 7.5, we describe the data structures that we use to implement our algorithm. On top of the standard data structures used to maintain the edge coloring χ, which allows us to implement Algorithms 2 and 3 efficiently, we also use data structures that allow us to efficiently maintain a

separable collection $\mathcal{U}$. More specifically, the data structures that we use to implement a separable collection $\mathcal{U}$ support the following queries.

- INSERT$_{\mathcal{U}}(\boldsymbol{g})$: The input to this query is a u-component $\boldsymbol{g}$. In response, the data structure adds $\boldsymbol{g}$ to $\mathcal{U}$ if $\mathcal{U} \cup \{\boldsymbol{g}\}$ is separable and outputs `fail` otherwise.
- DELETE$_{\mathcal{U}}(\boldsymbol{g})$: The input to this query is a u-component $\boldsymbol{g}$. In response, the data structure removes $\boldsymbol{g}$ from $\mathcal{U}$ if $\boldsymbol{g} \in \mathcal{U}$ and outputs `fail` otherwise.
- FIND-COMPONENT$_{\mathcal{U}}(x, c)$: The input to this query is a vertex $x \in V$ and a color $c \in [\Delta + 1]$. In response, the data structure returns the u-component $g \in \mathcal{U}$ with $c_{\boldsymbol{g}}(x) = c$ if such a u-component exists and outputs `fail` otherwise.
- MISSING-COLOR$_{\mathcal{U}}(x)$: The input to this query is a vertex $x \in V$. In response, the data structure returns an arbitrary color from the set $\mathsf{miss}_{\chi}(x) \setminus C_{\mathcal{U}}(x)$.[3]

The following claim shows that it is always possible to answer a MISSING-COLOR query.

Claim 7.1.5. *For each $x \in V$, the set $\mathsf{miss}_{\chi}(x) \setminus C_{\mathcal{U}}(x)$ is non-empty.*

Proof. Let d denote the number of uncolored edges incident on x. Since the collection $\mathcal{U}$ is separable, we have that $|C_{\mathcal{U}}(x)| \leq d$. Since $|\mathsf{miss}_{\chi}(x)| \geq d + 1$, it follows that $|C_{\mathcal{U}}(x)| < |\mathsf{miss}_{\chi}(x)|$ and so $\mathsf{miss}_{\chi}(x) \setminus C_{\mathcal{U}}(x) \neq \emptyset$. □

Furthermore, the data structure supports the following initialization operation.

- INITIALIZE(G, χ): Given a graph G and an edge coloring χ of G, we can initialize the data structure with an empty separable collection $\mathcal{U} = \emptyset$.

In Section 7.5, we show how to implement the initialization operation in $O(m)$ time and each of these queries in $O(1)$ time with the appropriate data structures. **These queries provide the 'interface' via which our algorithm will interact with the u-components.**

A Remark on Randomization. In order to get good space complexity and running time simultaneously, we implement the standard data structures (used for Algorithms 2 and 3) and the data structure supporting the preceding queries using *hashmaps* (see Section 7.5). The following proposition describes the hashmaps used by our data structures.[4]

Proposition 7.1.6 ([DadH90])**.** *There exists a dynamic dictionary that, given a parameter k, can handle k insertions, deletions, and map operations, uses $O(k)$ space and takes $O(1)$ worst-case time per operation with probability at least $1 - O(1/k^7)$.*

[3] Note that, since $|C_{\mathcal{U}}(x)| < |\mathsf{miss}_{\chi}(x)|$, such a color always exists.

[4] We could have even used the construction of [Kus22] to obtain exponentially small error probability in this case, but since we do not need this stronger guarantee, we stick with the simpler work of [DadH90].

This implementation gives us the guarantee that *across the run of the entire algorithm, every query made to one of these data structures takes* $O(1)$ *time and each initialization operation takes* $O(m)$ *time, with high probability.* Since the randomization used for the hashmaps is independent of the randomization used in the rest of the algorithm, **we implicitly condition on the high probability event that, every operation performed using a hashmap runs in** $O(1)$ **time throughout the rest of the chapter.**[5]

7.1.3 Vizing Fans in Separable Collections

Within our algorithm, we only construct Vizing fans and Vizing chains in a setting where there is some underlying separable collection $\mathcal{U}$. To ensure that activating and rotating colors around Vizing fans does not damage too many u-components, we choose the missing colors involved in Vizing fan constructions so that they 'avoid' the colors assigned to the u-components in $\mathcal{U}$.

Definition 7.1.7. *Let* $\mathcal{U}$ *be a separable collection and* $\mathbf{F} = (u, \alpha), (v_1, c_1), \ldots, (v_k, c_k)$ *be a Vizing fan. We say that the Vizing fan* $\mathbf{F}$ *is* $\mathcal{U}$-avoiding *if* $c_i \in \mathsf{miss}_\chi(v_i) \setminus C_{\mathcal{U}}(v_i)$ *for each leaf* $v_i \in \mathbf{F}$.

We say that a Vizing fan $\mathbf{F}$ *is a Vizing fan of the u-edge* $\boldsymbol{e} = (u, v, \alpha)$ if $\mathbf{F}$ is α-primed, has center u and its first leaf is v. The following lemma shows that we can always find a $\mathcal{U}$-avoiding Vizing fan for a u-edge.

Lemma 7.1.8. *Given a u-edge* $\boldsymbol{e} \in \mathcal{U}$*, there exists a* $\mathcal{U}$*-avoiding Vizing fan* $\mathbf{F}$ *of* $\boldsymbol{e}$*. Furthermore, we can compute such a Vizing fan in* $O(\Delta)$ *time.*

Proof. By Claim 7.1.5, we can always find a collection of colors $\{\mathsf{clr}(x)\}_{x \in V}$ such that $\mathsf{clr}(x) \in \mathsf{miss}_\chi(x) \setminus C_{\mathcal{U}}(x)$ for each $x \in V$. If we construct a Vizing fan $\mathbf{F}$ by calling `Vizing-Fan` and using such a collection of colors within the algorithm, then it follows that the Vizing fan $\mathbf{F}$ is $\mathcal{U}$-avoiding. By combining the standard implementation of Algorithm 2 with the queries described in Section 7.1.2 we can do this in $O(\Delta)$ time. □

The following lemma describes some crucial properties of $\mathcal{U}$-avoiding Vizing fans.

Lemma 7.1.9. *Let* χ *be a* $(\Delta+1)$*-edge coloring of a graph* G *and* $\mathcal{U}$ *be a separable collection. For any u-edge* $\boldsymbol{e} = (u, v, \alpha) \in \mathcal{U}$ *with a* $\mathcal{U}$*-avoiding Vizing fan* $\mathbf{F}$*, we have the following:*

1. *Rotating colors around* $\mathbf{F}$ *does not damage any u-component in* $\mathcal{U} \setminus \{\boldsymbol{e}\}$.
2. *Calling* $\mathsf{Vizing}(\mathbf{F})$ *damages at most one u-component in* $\mathcal{U} \setminus \{\boldsymbol{e}\}$*. Furthermore, we can identify the damaged u-component in* $O(1)$ *time.*

Proof. Let $\mathbf{F} = (u, \alpha), (v_1, c_1), \ldots, (v_k, c_k)$. Rotating colors around $\mathbf{F}$ will only remove the color c_i from the palette of a leaf v_i appearing in $\mathbf{F}$. Since $c_i \notin C_{\mathcal{U}}(v_i)$, these changes to the palettes will

[5] Alternatively, one can replace these hashmaps with balanced search trees to make these data structures deterministic, incurring $O(\log n)$ overhead in the running time of each operation.

not damage any u-component in $\mathcal{U}$. However, rotating colors around $\mathbf{F}$ will color the edge (u, v), damaging the u-edge $\boldsymbol{e}$.

Now, suppose we call Vizing($\mathbf{F}$). If the Vizing fan $\mathbf{F}$ is trivial, then the algorithm rotates all the colors in $\mathbf{F}$ and sets $\chi(u, v_k) \leftarrow c_k$. Rotating the colors can only damage $\boldsymbol{e} \in \mathcal{U}$, and removing c_k from the palette of u can damage at most one other u-component in $\mathcal{U}$ since $\mathcal{U}$ is separable. Using the queries from Section 7.1.2, we can check if such a u-component exist and return it in $O(1)$ time. If $\mathbf{F}$ is not trivial, then the algorithm flips the $\{\alpha, c_k\}$-alternating path $P = \texttt{Vizing-Path}(\mathbf{F})$ and rotates colors around $\mathbf{F}$. By similar arguments, rotating colors around $\mathbf{F}$ can only damage $\boldsymbol{e} \in \mathcal{U}$. Let x denote the endpoint of P that is not u. Flipping the path P might remove either the color α or c_k from the palette of x, which might damage at most one u-component in $\mathcal{U} \setminus \{e\}$ since $\mathcal{U}$ is separable. We can again identify this u-component in $O(1)$ time using the queries from Section 7.1.2. It also removes the color α from the palette of u, but this cannot damage any u-component other than $\boldsymbol{e}$ (again, since $\mathcal{U}$ is separable). □

7.2 The Main Algorithm

We are now ready to present our main technical result, which is a slightly weaker version of Theorem 7.0.1, and focuses on achieving a near-linear time algorithm for $(\Delta+1)$-edge coloring (instead of the exact $O(m \log n)$ time in Theorem 7.0.1; see the remark after that theorem). We will then use this theorem in Section 7.6 to conclude the proof of Theorem 7.0.1.

Theorem 7.2.1. *There is a randomized algorithm that, given any simple undirected graph $G = (V, E)$ on n vertices and m edges with maximum degree Δ, finds a $(\Delta+1)$-edge coloring of G in $\tilde{O}(m)$ time with high probability.*

Our main algorithm consists of two main components. The first component is an algorithm called Construct-U-Fans that takes a partial $(\Delta+1)$-edge coloring χ with λ uncolored edges and either extends χ to $\Omega(\lambda)$ of these edges or modifies the coloring to construct a separable collection of $\Omega(\lambda)$ u-fans. Lemma 7.2.2 summarizes the behavior of this algorithm.

Lemma 7.2.2. *Given a graph G, a partial $(\Delta+1)$-edge coloring χ of G and a set of λ uncolored edges U, the algorithm* Construct-U-Fans *does one of the following in $O(m + \Delta\lambda)$ time:*

1. *Extends the coloring to $\Omega(\lambda)$ uncolored edges.*
2. *Modifies χ to obtain a separable collection of $\Omega(\lambda)$ u-fans $\mathcal{U}$.*

The second component is an algorithm called Color-U-Fans that takes a collection of λ u-fans and extends the coloring to $\Omega(\lambda)$ of the edges in the u-fans. Lemma 7.2.3 summarizes the behavior of this algorithm. The reader may find it helpful to keep in mind that the algorithm for Lemma 7.2.3 is a generalization of algorithm for Lemma 4.2.2 in Section 4.2.

Lemma 7.2.3. *Given a graph G, a partial $(\Delta+1)$-edge coloring χ of G and a separable collection of λ u-fans $\mathcal{U}$, the algorithm* Color-U-Fans *extends χ to $\Omega(\lambda)$ edges in $O(m \log n)$ time w.h.p.*

In Sections 7.3 and 7.4, we prove Lemmas 7.2.2 and 7.2.3 respectively. Using these lemmas, we now show how to efficiently extend an edge coloring χ to the remaining uncolored edges.

Lemma 7.2.4. *Given a graph G and a partial $(\Delta+1)$-edge coloring χ of G with λ uncolored edges U, we can extend χ to the remaining uncolored edges in time $O((m+\Delta\lambda)\log^2 n)$ w.h.p.*

Proof. Let U denote the set of edges that are uncolored by χ. We can then apply Lemma 7.2.2 to either extend χ to a constant fraction of the edges in U or construct a separable collection of $\Omega(\lambda)$ u-fans $\mathcal{U}$ in $O(m+\Delta\lambda)$ time. In the second case, we can then apply Lemma 7.2.3 to color $\Omega(\lambda)$ of the edges contained in these u-fans in $O(m\log n)$ time w.h.p. Thus, we can extend χ to a constant fraction of the uncolored edges in $O(m\log n+\Delta\lambda)$ time w.h.p. After repeating this process $O(\log\lambda)\leq O(\log n)$ many times, no edges remain uncolored. Thus, we can extend the coloring χ to the entire graph in $O((m+\Delta\lambda)\log^2 n)$ time w.h.p. □

We now use this lemma to prove Theorem 7.2.1.

Proof of Theorem 7.2.1. We prove this by applying Lemma 7.2.4 to the standard Euler partition framework [GNK+85, Sin19]. Given a graph G, we partition it into two edge-disjoint subgraphs G_1 and G_2 on the same vertex set such that $\Delta(G_i)\leq\lceil\Delta/2\rceil$ for each G_i, where $\Delta(G_i)$ denotes the maximum degree of G_i. We then recursively compute a $(\Delta(G_i)+1)$-edge coloring χ_i for each G_i. Combining χ_1 and χ_2, we obtain a $(\Delta+3)$-edge coloring χ of G. We then uncolor the two smallest color classes in χ, which contain $O(m/\Delta)$ edges, and apply Lemma 7.2.4 to recolor all of the uncolored edges in χ using only $\Delta+1$ colors in $O(m\log^2 n)$ time w.h.p.

To show that the total running time of the algorithm is $O(m\log^3 n)$, first note that the depth of the recursion tree is $O(\log\Delta)$. Next, consider the i^{th} level of the recursion tree, for an arbitrary $i=O(\log\Delta)$: we have 2^i edge-disjoint subgraphs $G_1,\ldots,G_{2^i}$ such that $\Delta(G_j)\leq O(\Delta/2^i)$ and $\sum_{j=1}^{2^i}|E(G_j)|=m$. Since the total running time at recursion level i is $O(m\log^2 n)$ and the depth of the recursion tree is $O(\log\Delta)$, it follows that the total running time is $O(m\log^3 n)$ w.h.p. □

7.3 The Algorithm Construct-U-Fans: Proof of Lemma 7.2.2

As input, the algorithm Construct-U-Fans is given a graph G and a partial $(\Delta+1)$-edge coloring χ of G with λ uncolored edges. It begins by taking the λ uncolored edges in χ and using them to construct a separable collection $\mathcal{U}$ of λ u-edges in an obvious way, which we describe in Lemma 7.3.14. It then uses two subroutines, Prune-Vizing-Fans and Reduce-U-Edges, to reduce the number of u-edges in $\mathcal{U}$, either by coloring them or modifying χ to turn them into u-fans. More specifically, for each color $\alpha\in[\Delta+1]$, the algorithm considers the collection $\mathcal{E}_\alpha(\mathcal{U})$ of u-edges that are α-primed and (1) calls the subroutine Prune-Vizing-Fans to ensure that the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ have vertex-disjoint Vizing fans by either coloring the u-edges with overlapping Vizing fans or using them to create u-fans, and (2) calls the subroutine Reduce-U-Edges which either extends the coloring to the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ or uses them to create u-fans. Algorithm 7 gives the pseudocode for this algorithm.

Algorithm 7: Construct-U-Fans

```
Construct a separable collection of λ u-edges 𝒰
for each α ∈ [Δ + 1] do
    ℱ ← Prune-Vizing-Fans(𝒰, α)
    Reduce-U-Edges(𝒰, α, ℱ)
return 𝒰
```

Organization of Section 7.3. We first describe and analyze the subroutines Prune-Vizing-Fans and Reduce-U-Edges used by the algorithm Construct-U-Fans before proving Lemma 7.2.2 in Section 7.3.3.

7.3.1 The Subroutine Prune-Vizing-Fans

As input, the subroutine Prune-Vizing-Fans is given a graph G, a partial $(\Delta+1)$-edge coloring χ of G, a color α and a separable collection $\mathcal{U}$ with λ_α α-primed u-edges. We let $\mathcal{E}(\mathcal{U}) \subseteq \mathcal{U}$ denote the set of u-edges in $\mathcal{U}$ and $\mathcal{E}_\alpha(\mathcal{U}) \subseteq \mathcal{E}(\mathcal{U})$ denote the set of u-edges in $\mathcal{U}$ that are α-primed. The subroutine then modifies the coloring χ and collection $\mathcal{U}$ to ensure that the Vizing fans of the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ are vertex-disjoint.

Algorithm Description. The subroutine Prune-Vizing-Fans scans through the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ and constructs $\mathcal{U}$-avoiding Vizing fans at these u-edges (see Lemma 7.1.8), maintaining a vertex-disjoint subset $\mathcal{F}$ of these Vizing fans throughout the process. After constructing a Vizing fan $\mathbf{F}$ for a u-edge $\boldsymbol{e} \in \mathcal{E}_\alpha(\mathcal{U})$, the subroutine checks if $\mathbf{F}$ intersects some other fan in $\mathcal{F}$. If not, it adds $\mathbf{F}$ to $\mathcal{F}$. Otherwise, it uses these intersecting Vizing fans to either extend the coloring χ or construct a u-fan, removing the corresponding Vizing fans and u-edges from $\mathcal{F}$ and $\mathcal{U}$ respectively. See Figure 7.2 for an illustration.

More formally, let $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_\ell$ denote the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ at the time when the subroutine is called and let $\boldsymbol{e}_i = (u_i, v_i, \alpha)$. Initialize an empty set of Vizing fans $\mathcal{F} \leftarrow \emptyset$. Then, for each $i \in [\ell]$, the subroutine constructs a $\mathcal{U}$-avoiding Vizing fan $\mathbf{F}_i$ of $\boldsymbol{e}_i$ and checks if any of the vertices in $\mathbf{F}_i$ (i.e. any of the leaves or the center of $\mathbf{F}_i$) are contained in any of the Vizing fans in $\mathcal{F}$. Note that, since $\mathcal{U}$ is separable and $u_1, \ldots, u_\ell$ are the centers of α-primed u-edges, these vertices are all distinct. If $\mathbf{F}_i$ is vertex-disjoint from the Vizing fans in $\mathcal{F}$, we add $\mathbf{F}_i$ to $\mathcal{F}$. Otherwise, consider the following 3 cases.

- If u_i is a leaf in $\mathbf{F}_j$ for some $\mathbf{F}_j \in \mathcal{F}$: Then we rotate the fan $\mathbf{F}_j$ so that (u_i, u_j) is uncolored, set $\chi(u_i, u_j) \leftarrow \alpha$, remove $\mathbf{F}_j$ from $\mathcal{F}$ and remove $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{E}_\alpha(\mathcal{U})$.

If this is not the case, then some leaf of $\mathbf{F}_i$ appears in a Vizing fan in $\mathcal{F}$. Let w be the first such leaf in $\mathbf{F}_i$.

- If $w = u_j$ for some $\mathbf{F}_j \in \mathcal{F}$: Then we rotate the fan $\mathbf{F}_i$ so that (u_i, w) is uncolored, set $\chi(u_i, w) \leftarrow \alpha$, remove $\mathbf{F}_j$ from $\mathcal{F}$ and remove $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{E}_\alpha(\mathcal{U})$.

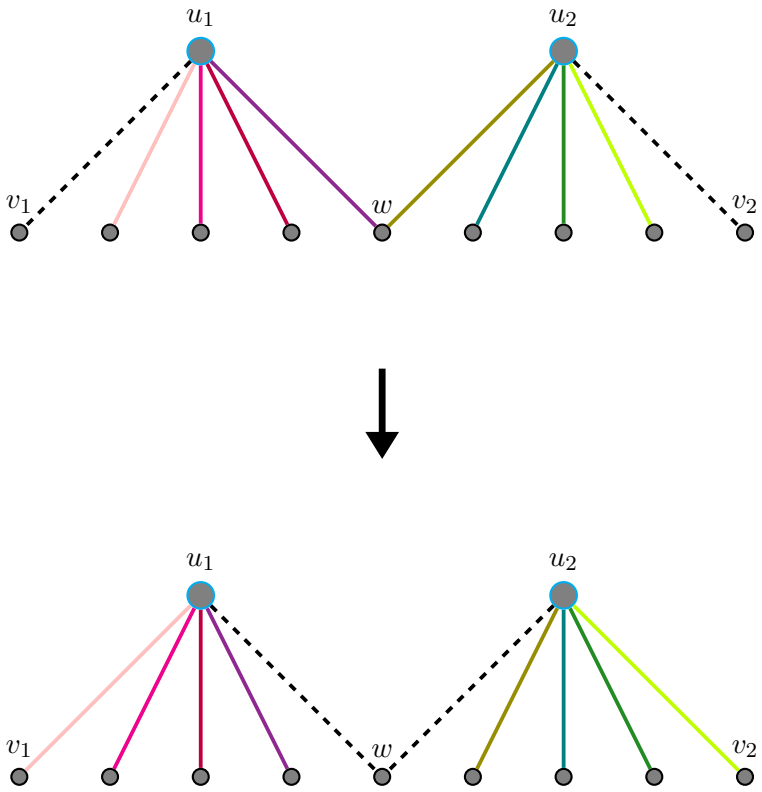

Figure 7.2: In this picture, we look at the Vizing fan around an uncolored edge (u_1, v_1), and it intersects with the Vizing fan of another edge (u_2, v_2) currently residing in $\mathcal{F}$ at vertex w. Then, we can rotate both Vizing fans and pair these two uncolored edges as a u-fan.

- If w is a leaf in $\mathbf{F}_j \in \mathcal{F}$: Then we rotate the fans $\mathbf{F}_i$ and $\mathbf{F}_j$ so that (u_i, w) and (u_j, w) are uncolored, create a u-fan $\boldsymbol{f} = (w, u_i, u_j, \beta, \alpha)$ for an arbitrary $\beta \in \mathsf{miss}_\chi(w) \setminus C_{\mathcal{U}}(w)$, remove $\mathbf{F}_j$ from $\mathcal{F}$, add $\boldsymbol{f}$ to $\mathcal{U}$ and remove $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{E}_\alpha(\mathcal{U})$.[6]

The subroutine then returns the set $\mathcal{F}$ of vertex-disjoint Vizing fans.

Analysis of Prune-Vizing-Fans

The following lemmas summarize the main properties of the subroutine Prune-Vizing-Fans.

Lemma 7.3.1. Prune-Vizing-Fans *returns a set $\mathcal{F}$ of vertex-disjoint $\mathcal{U}$-avoiding Vizing fans for all of the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$.*

Proof. We can show by induction that, after the subroutine finishes scanning the u-edge $\boldsymbol{e}_i$, the set $\mathcal{F}$ consists of vertex-disjoint $\mathcal{U}$-avoiding Vizing fans for the u-edges in $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_i\} \cap \mathcal{E}_\alpha(\mathcal{U})$.[7] This is true trivially for $i = 1$. Now, suppose that this is true for $i - 1$. If the $\mathcal{U}$-avoiding Vizing fan $\mathbf{F}_i$ of $\boldsymbol{e}_i$ does not intersect any of the Vizing fans in $\mathcal{F}$, then we add $\mathbf{F}_i$ to $\mathcal{F}$ and are done.

Thus, we assume this is not the case. Let x_0 denote u_i and $x_1, \ldots, x_k$ denote the leaves of the Vizing fan $\mathbf{F}_i$ (in order). Let x_p be the first vertex in the sequence $x_0, x_1, \ldots, x_k$ which appears in some Vizing fan $\mathbf{F}_j \in \mathcal{F}$. It follows from our algorithm that we remove $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{E}_\alpha(\mathcal{U})$ while only affecting the palettes of vertices in $\mathbf{F}_j$ and $x_0, \ldots, x_p$ by rotating the fans $\mathbf{F}_i$ and $\mathbf{F}_j$. Since all of the Vizing fans in $\mathcal{F}$ were vertex-disjoint, changing the palettes of vertices in $\mathbf{F}_j$ cannot affect the palettes of vertices in any of the Vizing fans in $\mathcal{F} \setminus \{\mathbf{F}_j\}$, and hence they are still valid. Similarly, none of the vertices $x_0, \ldots, x_p$ are contained in any of the Vizing fans in $\mathcal{F} \setminus \{\mathbf{F}_j\}$, so changing their palettes does not affect the validity of any other Vizing fans in $\mathcal{F}$. Thus, after removing $\mathbf{F}_j$ from $\mathcal{F}$, removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{E}_\alpha(\mathcal{U})$ and making the changes to χ, the set $\mathcal{F}$ consists of vertex-disjoint Vizing fans of the u-edges in $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_i\} \cap \mathcal{E}_\alpha(\mathcal{U})$. To see why all of these Vizing fans also remain $\mathcal{U}$-avoiding, note that any u-fan $\boldsymbol{f}$ added to $\mathcal{U}$ is vertex-disjoint from these Vizing fans by an analogous argument. □

Lemma 7.3.2. *The subroutine* Prune-Vizing-Fans *maintains the invariant that $\mathcal{U}$ is separable.*

Proof. We can show by induction that, after the subroutine finishes scanning the u-edge $\boldsymbol{e}_i$, the collection $\mathcal{U}$ is separable. This is true trivially for $i = 1$, since the subroutine does not change χ or $\mathcal{U}$ while scanning this u-edge. Now, suppose that this is true for $i - 1$. It follows from the inductive argument in Lemma 7.3.1 that the Vizing fans currently in $\mathcal{F}$ are all $\mathcal{U}$-avoiding. If we do not change χ or $\mathcal{U}$ while scanning $\boldsymbol{e}_i$, then we are done. Thus, we assume that this is not the case. The subroutine can modify χ and $\mathcal{U}$ in the following ways:

- Rotating colors around some $\mathbf{F}_j$ so that (u_j, u_i) is uncolored, setting $\chi(u_j, u_i) \leftarrow \alpha$ and removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$.

[6] Note that, since the vertices $u_1, \ldots, u_\ell$ are all distinct, these cases are exhaustive.

[7] Since we remove edges from $\mathcal{E}_\alpha(\mathcal{U})$ throughout this process, $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{i-1}\}$ might not be contained in $\mathcal{E}_\alpha(\mathcal{U})$.

- Rotating colors around $\mathbf{F}_i$ so that (u_i, u_j) is uncolored for some $\mathbf{F}_j \in \mathcal{F}$, setting $\chi(u_j, u_i) \leftarrow \alpha$ and removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$.
- Rotating colors around $\mathbf{F}_i$ and some $\mathbf{F}_j \in \mathcal{F}$ so that (u_i, u_j) is uncolored, removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$ and adding a u-fan $\boldsymbol{f}$ to $\mathcal{U}$.

In the first case, it follows from Lemma 7.1.9 (since the Vizing fans are all $\mathcal{U}$-avoiding) that rotating colors around $\mathbf{F}_j$ does not damage any u-component in $\mathcal{U} \setminus \{\boldsymbol{e}_j\}$. Furthermore, since $\mathcal{U}$ is separable, setting $\chi(u_j, u_i) \leftarrow \alpha$ does not damage any u-component in $\mathcal{U} \setminus \{\boldsymbol{e}_i, \boldsymbol{e}_j\}$. Since we remove $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$, we ensure that $\mathcal{U}$ remains separable. The same argument extends to the second case.

Finally, for the third case, it follows from Lemma 7.1.9 that rotating colors around $\mathbf{F}_i$ and $\mathbf{F}_j$ does not damage an u-component in $\mathcal{U} \setminus \{\boldsymbol{e}_i, \boldsymbol{e}_j\}$, so after removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$ we have that $\mathcal{U}$ is separable. Since we have that $\alpha \in C_{\mathcal{U}}(u_i) \cap C_{\mathcal{U}}(u_j)$ after removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$, we can see that $\boldsymbol{f}$ is indeed a u-fan and that $\mathcal{U} \cup \{\boldsymbol{f}\}$ is separable.[8] □

Lemma 7.3.3. *Each time* Prune-Vizing-Fans *modifies the coloring χ, it removes at most 2 u-edges from $\mathcal{E}_\alpha(\mathcal{U})$ and either (1) extends the coloring χ to one more edge, or (2) adds a u-fan to $\mathcal{U}$.*

Proof. This follows immediately from the description of the subroutine. □

The following lemma shows that this subroutine can be implemented efficiently.

Lemma 7.3.4. *We can implement* Prune-Vizing-Fans *to run in $O(\Delta\lambda_\alpha)$ time.*

Proof. By Lemma 7.1.8, we can construct each of the $\mathcal{U}$-avoiding Vizing fans $\mathbf{F}_i$ in $O(\Delta)$ time. As we scan through the u-edges $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_\ell$, we can maintain the set of vertices S that are contained in the Vizing fans in $\mathcal{F}$, along with pointers to the corresponding Vizing fan. This allows us to either verify that $\mathbf{F}_i$ is vertex-disjoint from the Vizing fans in $\mathcal{F}$ or to find the first vertex in $\mathbf{F}_i$ that is contained in one of these fans in $O(\Delta)$ time. We can then appropriately update the coloring χ and the collection $\mathcal{U}$ in $O(\Delta)$ time using the data structures outlined in Section 7.1.2. It follows that the entire subroutine takes $O(\Delta\lambda_\alpha)$ time. □

7.3.2 The Subroutine Reduce-U-Edges

As input, the subroutine Reduce-U-Edges is given a graph G, a partial $(\Delta+1)$-edge coloring χ of G, a color α, a separable collection $\mathcal{U}$ with λ_α α-primed u-edges and a set $\mathcal{F}$ of vertex-disjoint $\mathcal{U}$-avoiding Vizing fans for all of the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$. Let m_α denote the number of edges with color α in the input coloring χ. For each u-edge $\boldsymbol{e} \in \mathcal{E}_\alpha(\mathcal{U})$, let $\mathbf{F}_{\boldsymbol{e}}$ denote its Vizing fan in $\mathcal{F}$. Similarly, for each $\mathbf{F} \in \mathcal{F}$, let $\boldsymbol{e}_{\mathbf{F}}$ denote its corresponding u-edge in $\mathcal{E}_\alpha(\mathcal{U})$. The subroutine then either extends the coloring to the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ or uses them to construct u-fans.

Algorithm Description. The subroutine Reduce-U-Edges begins by activating all of the trivial Vizing fans in $\mathcal{F}$ and then removing them from $\mathcal{F}$. It then constructs Vizing chains at each of

[8] Note that, before removing $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ from $\mathcal{U}$, $c_{\boldsymbol{e}_i}(u_i) = c_{\boldsymbol{e}_j}(u_j) = \alpha$.

the remaining Vizing fans in $\mathcal{F}$ and explores the alternating paths $\texttt{Vizing-Path}(\mathbf{F})$ 'in parallel' (meaning, one edge at a time across all these paths). The subroutine continues this process until it either (1) reaches the end of one of these paths, or (2) two of these paths intersect. In the first case, it identifies the Vizing fan $\mathbf{F} \in \mathcal{F}$ whose Vizing chain is fully explored and calls $\mathsf{Vizing}(\mathbf{F})$ before removing $\mathbf{F}$ from $\mathcal{F}$. In the second case, it identifies the Vizing fans $\mathbf{F}, \mathbf{F}' \in \mathcal{F}$ corresponding to the intersecting Vizing chains and either extends the coloring χ to one more uncolored edge or creates a u-fan by shifting uncolored edges down these Vizing chains, before removing $\mathbf{F}$ and $\mathbf{F}'$ from $\mathcal{F}$. It repeats this process until at least half of the Vizing fans have been removed from $\mathcal{F}$.

More formally, the subroutine begins by scanning through all of the trivial Vizing fans $\mathbf{F} \in \mathcal{F}$ and calling $\mathsf{Vizing}(\mathbf{F})$, removing the $\mathbf{F}$ from $\mathcal{F}$ and $\boldsymbol{e}_{\mathbf{F}}$ from $\mathcal{U}$, and removing any other u-component damaged by calling $\mathsf{Vizing}(\mathbf{F})$ from $\mathcal{U}$ (see Lemma 7.1.9). Let $L \leftarrow 0$, $S \leftarrow \emptyset$ and $P_{\mathbf{F}} \leftarrow \emptyset$ for all $\mathbf{F} \in \mathcal{F}$. The subroutine then proceeds in *rounds* which consist of updating each of the paths $P_{\mathbf{F}}$ while maintaining the following invariant.

Invariant 7.3.5. *After updating a path $P_{\mathbf{F}}$, the following hold:*

1. *For each $\mathbf{F} \in \mathcal{F}$, $P_{\mathbf{F}}$ is the length $|P_{\mathbf{F}}|$ prefix of the alternating path $\texttt{Vizing-Path}(\mathbf{F})$, which we denote by $\texttt{Vizing-Path}(\mathbf{F})_{\leq |P_{\mathbf{F}}|}$.*
2. *The prefix paths $\{P_{\mathbf{F}}\}_{\mathbf{F} \in \mathcal{F}}$ are all edge-disjoint.*
3. *The set S is the union of all the edges in the prefix paths $\{P_{\mathbf{F}}\}_{\mathbf{F} \in \mathcal{F}}$.*
4. *The collection $\mathcal{U}$ is separable and $|\mathcal{F}| = |\mathcal{E}_\alpha(\mathcal{U})|$.*
5. *The Vizing fans in $\mathcal{F}$ are $\mathcal{U}$-avoiding and vertex-disjoint.*

The prefix paths $\{P_{\mathbf{F}}\}_{\mathbf{F} \in \mathcal{F}}$ maintained by the algorithm also satisfy the following invariant.

Invariant 7.3.6. *At the start of each round, we have that $|P_{\mathbf{F}}| = L$ for all $\mathbf{F} \in \mathcal{F}$.*

Immediately after starting a round, the subroutine increases the value of L by 1, invalidating Invariant 7.3.6. To restore this invariant, we need to update each of the prefix paths $P_{\mathbf{F}}$, which in turn may require us to update the other objects maintained by the subroutine to ensure that the other conditions of Invariant 7.3.5 are satisfied. The subroutine does this by scanning through each $\mathbf{F} \in \mathcal{F}$ and calling $\mathsf{Update\text{-}Path}(\mathbf{F})$, which we describe formally in Algorithm 8. The subroutine performs these rounds until $|\mathcal{F}| \leq \lambda_\alpha/2$, at which point it terminates.

Analysis of Reduce-U-Edges

The following lemmas summarize the main properties of the subroutine Reduce-U-Edges.

Lemma 7.3.7. *The subroutine* Reduce-U-Edges *satisfies Invariant 7.3.5.*

Algorithm 8: Update-Path($\mathbf{F}$)

1 $P_{\mathbf{F}} \leftarrow$ `Vizing-Path`$(\mathbf{F})_{\leq |P_{\mathbf{F}}|+1}$
2 Let (x, y) be the L^{th} edge in $P_{\mathbf{F}}$ and y be an endpoint of $P_{\mathbf{F}}$
3 **if** $(x, y) \in S$ **then**
4 Let $\mathbf{F}' \in \mathcal{F}$ be the Vizing fan such that $(x, y) \in P_{\mathbf{F}'}$
5 **if** (x, y) *appears in the same orientation in* $P_{\mathbf{F}}$ *and* $P_{\mathbf{F}'}$ **then**
6 Let (z, x) and (z', x) be the edges appearing before (x, y) in $P_{\mathbf{F}}$ and $P_{\mathbf{F}'}$ respectively
7 $\beta \leftarrow \chi(z, x)$
8 $\chi(z, x) \leftarrow \perp$ and $\chi(z', x) \leftarrow \perp$
9 Vizing($\mathbf{F}$)
10 Vizing($\mathbf{F}'$)
11 Remove $\mathbf{F}$ and $\mathbf{F}'$ from $\mathcal{F}$ and $\boldsymbol{e}_{\mathbf{F}}$ and $\boldsymbol{e}_{\mathbf{F}'}$ from $\mathcal{U}$
12 Add the u-fan $\boldsymbol{f} = (x, z, z', \beta, \alpha)$ to $\mathcal{U}$
13 **else**
14 $\chi(x, y) \leftarrow \perp$
15 Vizing($\mathbf{F}$)
16 Vizing($\mathbf{F}'$)
17 Remove $\mathbf{F}$ and $\mathbf{F}'$ from $\mathcal{F}$ and $\boldsymbol{e}_{\mathbf{F}}$ and $\boldsymbol{e}_{\mathbf{F}'}$ from $\mathcal{U}$
18 $S \leftarrow S \setminus (P_{\mathbf{F}} \cup P_{\mathbf{F}'})$
19 **return**
20 $S \leftarrow S \cup \{(x, y)\}$
21 **if** $P_{\mathbf{F}}$ *is maximal* **then**
22 Vizing($\mathbf{F}$)
23 Remove $\mathbf{F}$ from $\mathcal{F}$ and $\boldsymbol{e}_{\mathbf{F}}$ from $\mathcal{U}$
24 $S \leftarrow S \setminus P_{\mathbf{F}}$
25 **if** *there exists* $\boldsymbol{g} \in \mathcal{U}$ *with* $c_{\boldsymbol{g}}(y) = \chi(x, y)$ **then**
26 Remove $\boldsymbol{g}$ from $\mathcal{U}$ (see Lemma 7.1.9)
27 **if** *there exists* $\mathbf{F}' \in \mathcal{F}$ *such that* $y \in \mathbf{F}$ **then**
28 Remove $\mathbf{F}'$ from $\mathcal{F}$ and $\boldsymbol{e}_{\mathbf{F}'}$ from $\mathcal{U}$ // $\boldsymbol{e}_{F'}$ might also get removed in Line 26
29 $S \leftarrow S \setminus P_{\mathbf{F}'}$

Proof. We show that, as long as Invariant 7.3.5 is satisfied, calling Update-Path($\mathbf{F}$) for some $\mathbf{F} \in \mathcal{F}$ maintains the invariant. We first note that Invariant 7.3.5 is trivially satisfied immediately after initializing L, S and $\{P_{\mathbf{F}}\}_{\mathbf{F}\in\mathcal{F}}$.

After activating and removing all of the trivial Vizing fans in $\mathcal{F}$, Conditions 1-3 are clearly still satisfied (since the paths are all empty). For Condition 4, note that the subroutine removes any damaged u-components from $\mathcal{U}$, removes $\boldsymbol{e}$ from $\mathcal{E}_\alpha(\mathcal{U})$ if it removes $\mathbf{F}_{\boldsymbol{e}}$ from $\mathcal{F}$, and that activating some trivial $\mathbf{F}_{\boldsymbol{e}} \in \mathcal{F}$ cannot damage any $\boldsymbol{e}' \in \mathcal{E}_\alpha(\mathcal{U}) \setminus \{\boldsymbol{e}\}$ since their Vizing fans are vertex-disjoint. For Condition 5, note that the subroutine does not add any u-component to $\mathcal{U}$, so the vertex-disjoint Vizing fans remain $\mathcal{U}$-avoiding. This establishes the base case of the induction.

Now, assume that Invariant 7.3.5 is satisfied and suppose that we call Update-Path($\mathbf{F}$) for some $\mathbf{F} \in \mathcal{F}$. We now argue that each condition of Invariant 7.3.5 is satisfied after handling this call.

Conditions 2 and 3: After extending $P_{\mathbf{F}}$ by one more edge, we check if there is some other path $P_{\mathbf{F}'}$ that intersects the updated path $P_{\mathbf{F}}$ at this new edge. If so, we remove both $\mathbf{F}$ and $\mathbf{F}'$ from $\mathcal{F}$, ensuring that the remaining paths are edge-disjoint. Otherwise, the paths are all edge-disjoint. It's straightforward to verify that S is updated correctly in each case.

Condition 4: We first show that the collection $\mathcal{U}$ remains separable. If this call to Update-Path does not change χ or $\mathcal{U}$, then clearly $\mathcal{U}$ remains separable. Note that the Vizing fans in $\mathcal{F}$ are all $\mathcal{U}$-avoiding and recall Lemma 7.1.9. We now consider the following three cases.

(1) If we enter the **if** statement on Line 5, then calling Vizing($\mathbf{F}$) and Vizing($\mathbf{F}'$) after uncoloring edges on their Vizing chains does not damage any u-component in $\mathcal{U}$ apart from $\boldsymbol{e}_{\mathbf{F}}$ and $\boldsymbol{e}_{\mathbf{F}'}$, which are both removed from $\mathcal{U}$. This is because shifting colors around the Vizing fans $\mathbf{F}$ and $\mathbf{F}'$ does not damage any u-components apart from $\boldsymbol{e}_{\mathbf{F}}$ and $\boldsymbol{e}_{\mathbf{F}'}$ (see Lemma 7.1.9) and truncating an alternating path before flipping it ensures that we do not remove any colors from the palettes of the vertices on that path. Finally, note that the u-fan $\boldsymbol{f}$ that we add to $\mathcal{U}$ only uses colors that were previously unavailable. See the example Figure 7.3 for an illustration.

(2) If we enter the **else** statement on Line 13, then the argument is completely analogous to the previous case, except that we do not add any u-components to $\mathcal{U}$ (see Lemma 7.1.9). See Figure 7.4 for an illustration.

(3) If we enter the **if** statement on Line 21, then calling Vizing($\mathbf{F}$) damages at most one u-component in $\mathcal{U}$ apart from $\boldsymbol{e}_{\mathbf{F}}$ (see Lemma 7.1.9). In particular, if there is such a u-component $\boldsymbol{g}$, it will contain the vertex y and have $c_{\boldsymbol{g}}(x) = \chi(x, y)$. Thus, we check if such a u-component exists and remove it from $\mathcal{U}$.

Finally, to see that $|\mathcal{F}| = |\mathcal{E}_\alpha(\mathcal{U})|$, we can verify that we remove $\boldsymbol{e}$ from $\mathcal{E}_\alpha(\mathcal{U})$ if and only if we remove $\mathbf{F}_{\boldsymbol{e}}$ from $\mathcal{F}$. It's clear that whenever we remove some $\mathbf{F}'$ from $\mathcal{F}$, we also remove $\boldsymbol{e}_{\mathbf{F}'}$ from $\mathcal{E}_\alpha(\mathcal{U})$. Similarly, if we remove a u-edge $\boldsymbol{e}$ from $\mathcal{E}_\alpha(\mathcal{U})$ in Line 26, then we can see that we then remove $\mathbf{F}_{\boldsymbol{e}}$ from $\mathcal{F}$ in Line 28 immediately afterwards.

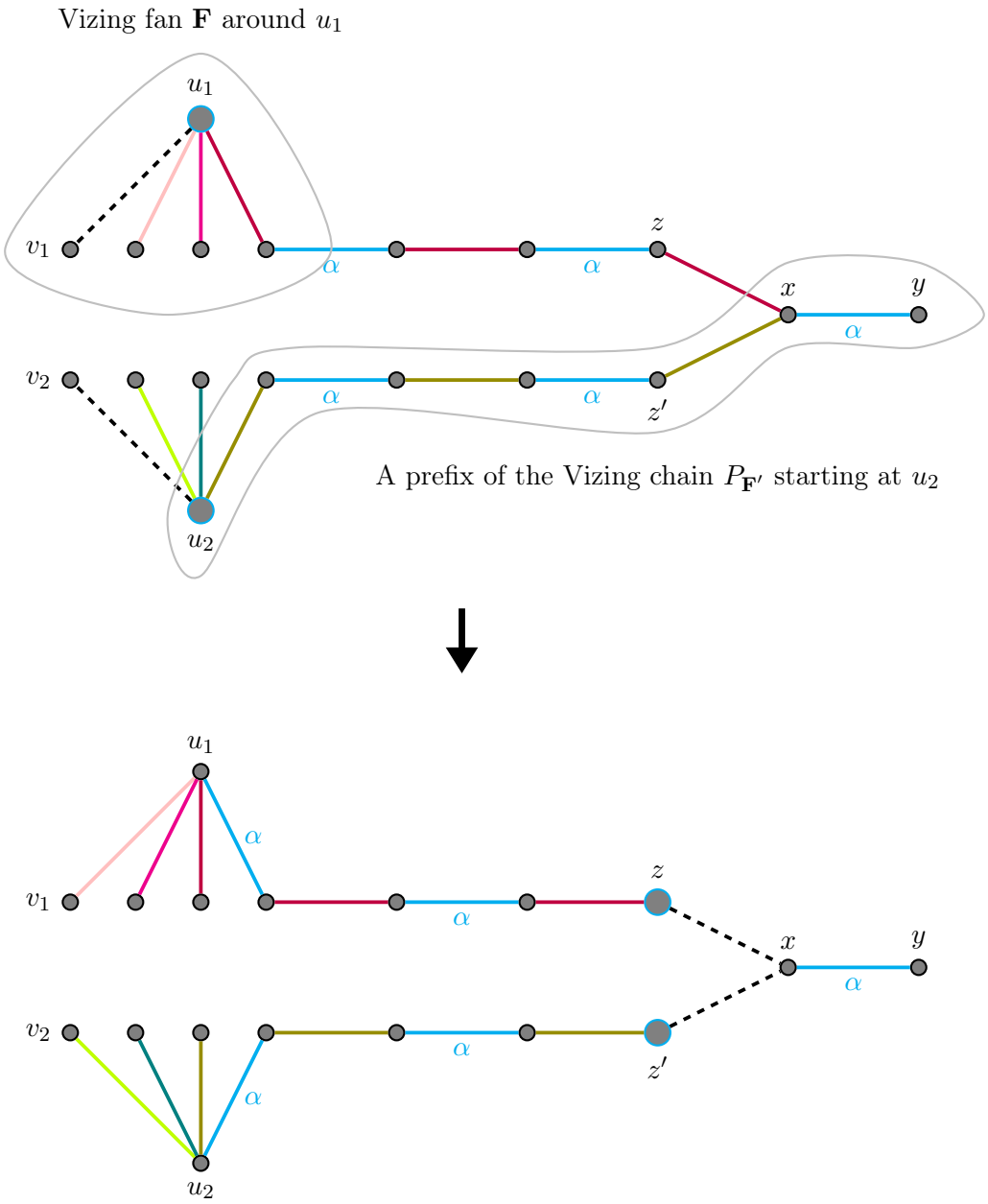

Figure 7.3: In this picture, α is blue, and two uncolored edges $\boldsymbol{e}_{\mathbf{F}} = (u_1, v_1), \boldsymbol{e}_{\mathbf{F}'} = (u_2, v_2)$ generate Vizing fans $\mathbf{F}$ and $\mathbf{F}'$, and the two Vizing chains intersect at (x, y) for the first time and in the same direction. Then, we can rotate both Vizing fans and partially flip $P_{\mathbf{F}}$ and $P_{\mathbf{F}'}$ to make a u-fan.

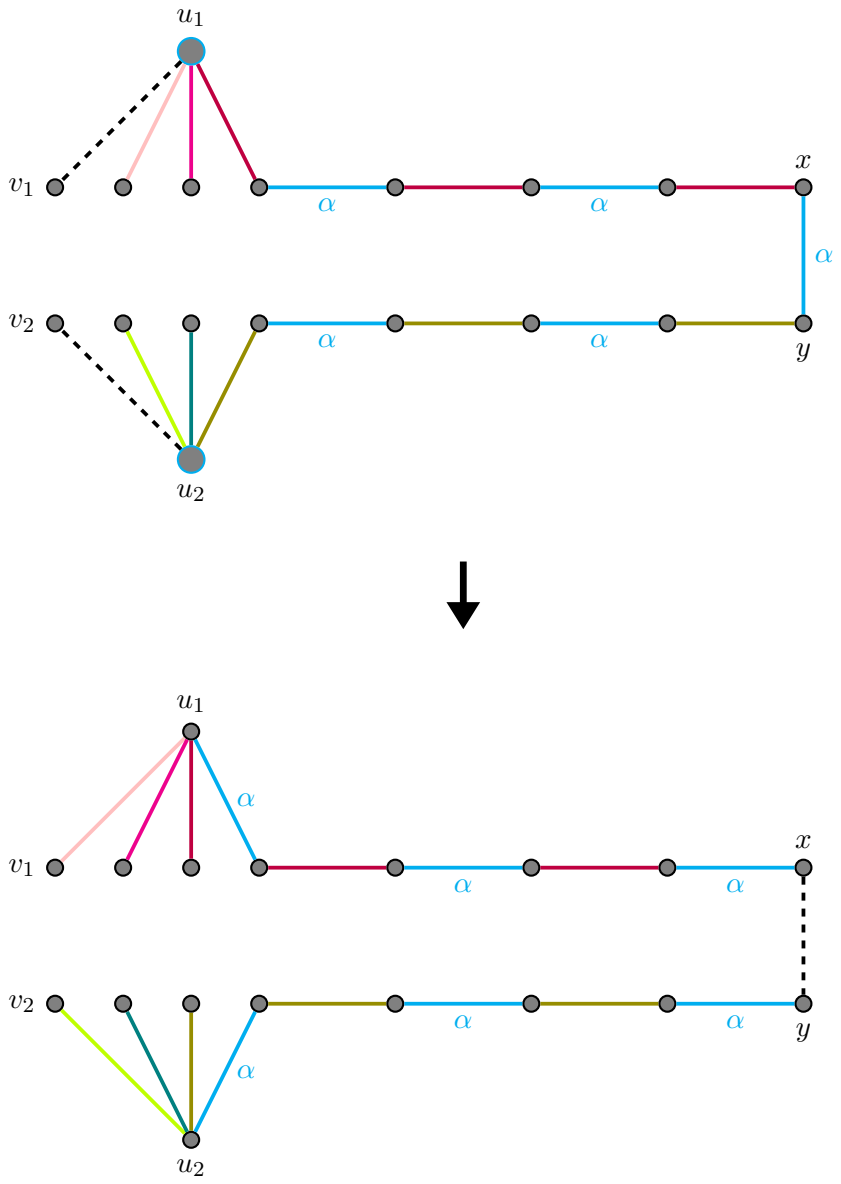

Figure 7.4: In this picture, α is blue, and two uncolored edges $\boldsymbol{e}_{\mathbf{F}} = (u_1, v_1), \boldsymbol{e}_{\mathbf{F}'} = (u_2, v_2)$ generate Vizing fans $\mathbf{F}$ and $\mathbf{F}'$, and the two Vizing chains intersect at (x, y) for the first time and in the opposite direction. Then, we can uncolor (x, y) and color both $\boldsymbol{e}_{\mathbf{F}}, \boldsymbol{e}_{\mathbf{F}'}$.

Condition 5: Let $\mathbf{F}'$ be some Vizing fan that remains in $\mathcal{F}$ after handling this call. Then, for any Vizing fan $\mathbf{F}''$ that was activated during the call, we know that its corresponding Vizing chain did *not* end at $\mathbf{F}'$, otherwise $\mathbf{F}'$ would have been removed from $\mathcal{F}$ (see Line 29). Thus, activating $\mathbf{F}''$ (which is vertex-disjoint from $\mathbf{F}'$) does not change the palettes of any vertices in $\mathbf{F}'$. Note that, since the center of $\mathbf{F}'$ is missing α, it is not possible for the Vizing chain of $\mathbf{F}''$ to contain any edges in $\mathbf{F}'$ without ending at $\mathbf{F}'$. Hence, the Vizing fans remaining in $\mathcal{F}$ are still valid Vizing fans and remain vertex-disjoint. To see that they also remain $\mathcal{U}$-avoiding, note that if we add a u-fan $\boldsymbol{f}$ to $\mathcal{U}$ (see Line 12), the color that $\boldsymbol{f}$ assigns to a vertex $x \in \boldsymbol{f}$ was previously not available at x before the start of the call to Update-Path. Thus, if x is contained in the Vizing fan $\mathbf{F}'$, its color within $\mathbf{F}'$ is not $c_{\boldsymbol{f}}(x)$.

Condition 1: Let $\mathbf{F}'$ be some Vizing fan that remains in $\mathcal{F}$ after handling this call. We now show that the first $|P_{\mathbf{F}'}|$ edges of `Vizing-Path`$(\mathbf{F}')$ do not change throughout this call. It follows from the argument for Condition 5 that neither the colors of the edges nor the palettes of the vertices in $\mathbf{F}'$ change during this call. It remains to show that none of the edges in $P_{\mathbf{F}'}$ change color during this call. We can see that $P_{\mathbf{F}'}$ and $P_{\mathbf{F}}$ are edge-disjoint since otherwise $\mathbf{F}'$ would have been removed from $\mathcal{F}$. Consequently, $P_{\mathbf{F}'}$ is edge-disjoint from any Vizing chain that is activated during the call, and hence none of the edges in $P_{\mathbf{F}'}$ change colors. □

Lemma 7.3.8. *The subroutine* Reduce-U-Edges *satisfies Invariant 7.3.6.*

Proof. This follows from the fact that initially $|P_{\mathbf{F}}| = 0$ for all $\mathbf{F} \in \mathcal{F}$ and that the subroutine calls Update-Path$(\mathbf{F})$ for each $\mathbf{F} \in \mathcal{F}$ during each round, which increases $|P_{\mathbf{F}}|$ by 1. □

Lemma 7.3.9. *Each time* Reduce-U-Edges *modifies the coloring χ, it removes at most 2 u-edges from $\mathcal{E}_\alpha(\mathcal{U})$ and either (1) extends the coloring χ to one more edge and removes at most one other u-component from $\mathcal{U}$, or (2) adds a u-fan to $\mathcal{U}$.*

Proof. While activating some trivial Vizing fan $\mathbf{F}_{\boldsymbol{e}} \in \mathcal{F}$ at the start, we remove $\boldsymbol{e}$ from $\mathcal{U}$ (and thus also from $\mathcal{E}_\alpha(\mathcal{U})$) along with at most one other u-component in $\mathcal{U}$ that is damaged by this operation (see Lemma 7.1.9). Now, suppose that the subroutine calls Update-Path$(\mathbf{F})$ for some $\mathbf{F} \in \mathcal{F}$. By considering each case, we can verify that the subroutine either makes no changes to χ and $\mathcal{U}$ or it removes at most 2 u-edges from $\mathcal{E}_\alpha(\mathcal{U})$ and either (1) extends the coloring χ to one more edge (see Lines 12-14 and Line 22) and removes at most one other u-component from $\mathcal{U}$ (see Line 26), or (2) adds a u-fan to $\mathcal{U}$ (see Line 12). □

Running Time. We now show how to implement this subroutine and analyse its running time. We begin with the following claim which shows that Update-Path can be implemented efficiently.

Claim 7.3.10. *Each call to* Update-Path$(\mathbf{F})$ *for some $\mathbf{F} \in \mathcal{F}$ that does not remove $\mathbf{F}$ from $\mathcal{F}$ takes $O(1)$ time. Otherwise, it takes $O(\Delta + L)$ time.*

Proof. By Invariants 7.3.5 and 7.3.6, we know that $P_{\mathbf{F}} = \texttt{Vizing-Path}(\mathbf{F})_{\leq L}$ when the subroutine calls Update-Path($\mathbf{F}$). Thus, updating $P_{\mathbf{F}}$ to $\texttt{Vizing-Path}(\mathbf{F})_{\leq L+1}$ only requires computing the next edge in the path, which can be done in $O(1)$ time using our data structures (see Section 7.5).

If this call does not remove $\mathbf{F}$ from $\mathcal{F}$, then we know that $P_{\mathbf{F}}$ is not maximal and is also edge-disjoint from the other prefix paths maintained by the subroutine. In this case, we update S in $O(1)$ time and do not modify χ or $\mathcal{U}$.

On the other hand, if this call does remove $\mathbf{F}$ from $\mathcal{F}$, then we need to activate $O(1)$ many Vizing chains of length $O(L)$, which can be done in $O(\Delta + L)$ time. Removing the edges of the corresponding paths from S can also be done in $O(L)$ time. Finally, using the data structures outlined in Section 7.1.2, we can update $\mathcal{F}$ and $\mathcal{U}$ in $O(1)$ time. □

Let $\mathcal{F}^\star$ denote the subset of Vizing fans that get removed from $\mathcal{F}$ by the subroutine Reduce-U-Edges before it terminates. For each $\mathbf{F} \in \mathcal{F}^\star$, let $L_{\mathbf{F}}$ denote the value of L at the time that $\mathbf{F}$ is removed from $\mathcal{F}$.

Claim 7.3.11. *The total time spent handling calls to* Update-Path *is at most*

$$O(\Delta\lambda_\alpha) + O(1) \cdot \sum_{\mathbf{F}\in\mathcal{F}^\star} L_{\mathbf{F}}. \tag{7.1}$$

Proof. Let $\mathbf{F} \in \mathcal{F}^\star$. We can observe that the subroutine calls Update-Path($\mathbf{F}$) at most $L_{\mathbf{F}}$ times. It follows from Claim 7.3.10 that the last call takes $O(\Delta + L_{\mathbf{F}})$ time while the rest take $O(1)$ time. Thus, the total time spent handling calls to Update-Path($\mathbf{F}$) is at most $O(\Delta + L_{\mathbf{F}})$. Summing over each $\mathbf{F} \in \mathcal{F}^\star$, we get that the total time spent handling calls to Update-Path is at most

$$\sum_{\mathbf{F}\in\mathcal{F}^\star} O(\Delta + L_{\mathbf{F}}) \leq O(\Delta\lambda_\alpha) + O(1) \cdot \sum_{\mathbf{F}\in\mathcal{F}^\star} L_{\mathbf{F}}.$$ □

Recall that m_α denotes the number of edges with color α when we first call the subroutine.

Claim 7.3.12. *For each $\mathbf{F} \in \mathcal{F}^\star$, we have that $L_{\mathbf{F}} \leq O(m_\alpha/\lambda_\alpha)$.*

Proof. Let $L_{\max}$ denote the value of L at the start of the final round performed by the subroutine. We now show that $L_{\max} \leq 8(m_\alpha + 4)/\lambda_\alpha$, which implies the claim. At the start of the final round, we know that $|\mathcal{F}| > \lambda_\alpha/2$, otherwise the subroutine would terminate. Furthermore, it follows from Invariants 7.3.5 and 7.3.6 that, at the start of this round, the alternating paths $\{P_{\mathbf{F}}\}_{\mathbf{F}\in\mathcal{F}}$ form a collection of at least $\lambda_\alpha/2$ edge-disjoint $\{\alpha, \cdot\}$-alternating paths of length $L_{\max}$ in G. Let T denote the total length of these paths. We can observe that T is at most $3m'_\alpha$, where m'_α is the number of edges that currently have color α, since at least a third of the edges in each of these paths has color α (note that any $\{\alpha, \cdot\}$-alternating path of length k has at least $\lfloor k/2 \rfloor$ edges with color α). We can also observe that $m'_\alpha \leq m_\alpha + 2\lambda_\alpha$ since the number of edges with color α increases by at most 2 each time we activate a Vizing fan in $\mathcal{F}$. Thus, it follows that

$$\frac{\lambda_\alpha}{2} \cdot L_{\max} \leq T \leq 3m'_\alpha \leq 3(m_\alpha + 2\lambda_\alpha),$$

and so $L_{\max} \leq 6m_\alpha/\lambda_\alpha + 12$. □

Lemma 7.3.13. *We can implement* Reduce-U-Edges *to run in* $O(m_\alpha + \Delta\lambda_\alpha)$ *time.*

Proof. Activating the trivial Vizing fans in $\mathcal{F}$ when the subroutine is first called can be done in $O(\Delta\lambda_\alpha)$ time. The running time of the rest of the subroutine is dominated by the time taken to handle the call to Update-Path. Combining Claims 7.3.11 and 7.3.12, it follows that the total time spent handling calls to Update-Path is at most

$$O(\Delta\lambda_\alpha) + O(1) \cdot \sum_{\mathbf{F} \in \mathcal{F}^\star} L_{\mathbf{F}} \leq O(\Delta\lambda_\alpha) + O(\lambda_\alpha) \cdot O\left(\frac{m_\alpha}{\lambda_\alpha}\right) \leq O(m_\alpha + \Delta\lambda_\alpha).$$ □

7.3.3 Analysis of Construct-U-Fans: Proof of Lemma 7.2.2

The algorithm begins by constructing a separable collection of λ u-components $\mathcal{U}$. The following lemma shows that this can be done efficiently.

Lemma 7.3.14. *Given a graph* G *and a partial* $(\Delta+1)$*-edge coloring* χ *of* G *with* λ *uncolored edges, we can construct a separable collection of* λ *u-edges* $\mathcal{U}$ *in* $O(m)$ *time.*

Proof. The algorithm first initializes an empty separable collection $\mathcal{U} = \emptyset$. It then scans through the edges of the graph and retrieves the λ edges $e_1, \ldots, e_\lambda$ that are uncolored by χ. The algorithm then scans through each of these uncolored edges and, for each $e_i = (u_i, v_i)$, picks a missing color $\alpha_i \in \mathsf{miss}_\chi(u_i) \setminus C_{\mathcal{U}}(u_i)$ (see Claim 7.1.5) and adds the u-edge $\boldsymbol{e}_i := (u_i, v_i, \alpha_i)$ to $\mathcal{U}$. It's easy to verify that the resulting collection $\mathcal{U}$ is separable and contains λ u-edges. Using the data structures outlined in Section 7.1.2, finding such a color and adding a u-edge to $\mathcal{U}$ can be done in $O(1)$ time. Thus, this entire process can be implemented in $O(m)$ time. □

Let m_α denote the number of edges that have color α w.r.t. the initial coloring χ when we first call Construct-U-Fans. The following claim shows that we cannot create too many more edges with the color α throughout this sequence of calls to the subroutines Prune-Vizing-Fans and Reduce-U-Edges for each color $\alpha \in [\Delta+1]$.

Claim 7.3.15. *For each color* $\alpha \in [\Delta+1]$*, we have that at most* $m_\alpha + O(\lambda)$ *edges have color* α *throughout the entire run of the algorithm* Construct-U-Fans.

Proof. This follows from the fact that the number of edges with color α can only increase when we extend the coloring χ to some uncolored edge. Furthermore, it can only increase by at most 2 every time this happens. Thus, for any $\alpha \in [\Delta+1]$, the number of edges with color α can increase by at most λ throughout the entire run of the algorithm Construct-U-Fans. □

Let λ_α denote the number of α-primed u-edges in the initial collection $\mathcal{U}$. Note that the number of such u-edges can only decrease throughout the run of this algorithm. It follows from Lemmas 7.3.4

and 7.3.13 that the total running time of the algorithm Construct-U-Fans across all of these calls to the subroutines Prune-Vizing-Fans and Reduce-U-Edges is at most

$$\sum_{\alpha \in [\Delta+1]} (O(\lambda_\alpha \Delta) + O(m_\alpha + \lambda + \lambda_\alpha \Delta)) \leq O(m + \Delta\lambda).$$

Lemma 7.3.16. *The algorithm* Construct-U-Fans *either extends the coloring χ to at least $\lambda/18$ more edges or returns a separable collection of at least $\lambda/18$ u-fans $\mathcal{U}$.*

Proof. To see why this lemma holds, consider the following three quantities and how they evolve over time throughout the execution of the algorithm Construct-U-Fans: The number of edges that the algorithm has extended the coloring to so far, $\Psi_{\mathtt{c}}$, the number of u-edges in $\mathcal{U}$, $\Psi_{\mathtt{e}} := |\mathcal{E}|$, and the number of u-fans in $\mathcal{U}$, $\Psi_{\mathtt{f}} := |\mathcal{U} \setminus \mathcal{E}|$. Immediately after constructing the separable collection of λ u-edges $\mathcal{U}$, it holds that $\Psi_{\mathtt{c}} = 0$, $\Psi_{\mathtt{e}} = \lambda$ and $\Psi_{\mathtt{f}} = 0$. We next argue that by the time the execution of the algorithm has finished, either $\Psi_{\mathtt{c}} \geq \lambda/18$ or $\Psi_{\mathtt{f}} \geq \lambda/18$ must hold. To this end, we employ the following potential function argument.

Claim 7.3.17. *The quantity $3(\Psi_{\mathtt{f}} + 2\Psi_{\mathtt{c}}) + \Psi_{\mathtt{e}}$ is non-decreasing throughout the algorithm's execution.*

Proof. Consider first a call made by algorithm Construct-U-Fans to the Prune-Vizing-Fans subroutine. By Lemma 7.3.3, the subroutine repeatedly removes at most 2 u-edges from $\mathcal{U}$ and either (1) adds a u-fan to $\mathcal{U}$, or (2) extends the coloring to another edge. This may decrease $\Psi_{\mathtt{e}}$ by at most 2 while increasing one of $\Psi_{\mathtt{f}}$ or $\Psi_{\mathtt{c}}$ by at least 1, so $3(\Psi_{\mathtt{f}} + 2\Psi_{\mathtt{c}}) + \Psi_{\mathtt{e}}$ increases in this case.

Next, consider a call to the Reduce-U-Edges subroutine. By Lemma 7.3.9, during any iteration, the subroutine removes at most 3 u-edges from $\mathcal{U}$ and either (1) adds a u-fan to $\mathcal{U}$, or (2) extends the coloring to another edge while removing at most one u-fan from $\mathcal{U}$. This may decrease $\Psi_{\mathtt{e}}$ by at most 3 while increasing $\Psi_{\mathtt{f}} + 2\Psi_{\mathtt{c}}$ by at least 1, so $3(\Psi_{\mathtt{f}} + 2\Psi_{\mathtt{c}}) + \Psi_{\mathtt{e}}$ cannot decrease in value. □

Immediately after constructing the separable collection of λ u-edges in $\mathcal{U}$, we have $3(\Psi_{\mathtt{f}}+2\Psi_{\mathtt{c}})+\Psi_{\mathtt{e}} = \lambda$. Claim 7.3.17 implies that $3(\Psi_{\mathtt{f}} + 2\Psi_{\mathtt{c}}) + \Psi_{\mathtt{e}} \geq \lambda$ holds at all times afterwards. Since we have that $|\mathcal{E}| \leq \lambda/2$ when the algorithm terminates, we know that $\Psi_{\mathtt{e}} \leq \lambda/2$ at this time, so at that moment we have $3(\Psi_{\mathtt{f}} + 2\Psi_{\mathtt{c}}) \geq \lambda/2$. Hence, either $\Psi_{\mathtt{f}} \geq \lambda/18$ or $\Psi_{\mathtt{c}} \geq \lambda/18$ must hold. □

7.4 The Algorithm Color-U-Fans: Proof of Lemma 7.2.3

As input, the algorithm Color-U-Fans is given a graph G, a partial $(\Delta+1)$-edge coloring χ of G, and a collection of separable u-fans $\mathcal{U}$ of size λ. It then uses two further subroutines, Prime-U-Fans and Activate-U-Fans, to prime u-fans with the same colors and then activate them. More specifically, the algorithm first identifies the two least common colors $\alpha, \beta \in [\Delta + 1]$. It calls the subroutine Prime-U-Fans which proceeds to prime $\Omega(\lambda/\Delta)$ of the u-fans in $\mathcal{U}$ with the colors α and β (i.e. to modify the coloring χ and these u-fans so that they are $\{\alpha, \beta\}$-primed). It then calls Activate-U-Fans

which extends the coloring χ to $\Omega(\lambda/\Delta)$ of the edges in these u-fans. The algorithm repeats this process until it has colored $\Omega(\lambda)$ edges. Algorithm 9 gives the pseudocode for Color-U-Fans.

Algorithm 9: Color-U-Fans($\mathcal{U}$)

```
for Δ/2 iterations do
    Let α, β ∈ [Δ + 1] be the two least common colors in χ
    Prime-U-Fans(U, α, β)
    Activate-U-Fans(U, α, β)
```

Organization of Section 7.4. We begin by describing and analyzing the subroutines Prime-U-Fans and Activate-U-Fans used by Color-U-Fans before proving Lemma 7.2.3 in Section 7.4.3.

7.4.1 The Subroutine Prime-U-Fans

As input, this subroutine is given a graph G, a partial $(\Delta+1)$-edge coloring χ of G, a collection of separable u-fans $\mathcal{U}$ of size λ, and the two least common colors $\alpha, \beta \in [\Delta+1]$ (see Claim 7.4.1). It then proceeds in *iterations*, where in each iteration it samples a u-fan $\boldsymbol{f}$ from $\mathcal{U}$ uniformly at random and attempts to prime $\boldsymbol{f}$ with the colors α and β. The subroutine maintains a subset $\Phi \subseteq \mathcal{U}$ of $\{\alpha,\beta\}$-primed u-fans. The subroutine performs iterations until $|\Phi| = \Omega(\lambda/\Delta)$, after which we proceed to call the subroutine Activate-U-Fans to extend the coloring χ to edges in the u-fans in Φ. The pseudocode in Algorithm 10 gives a formal description of the subroutine Prime-U-Fans.

Algorithm 10: Prime-U-Fans($\mathcal{U}$)

```
Φ ← ∅ and λ ← |U|
while |Φ| < λ/(48Δ) do
    Sample f = (u, v, w, γ, δ) ~ U independently and u.a.r.
    (α', β') ← (α, β)        // α' and β' ensure we never flip an {α, β}-alt. path
    if γ = β or δ = α then
        (α', β') ← (β, α)
    Let P_u be the {α', γ}-alternating path starting at u
    Let P_v and P_w be the {β', δ}-alternating paths starting at v and w respectively
    cost(f) := |P_u| + |P_v| + |P_w|
    if cost(f) ≤ 128m/λ then
        Let S denote the set of endpoints of P_u, P_v and P_w
        if the vertices in S ∪ {u, v, w} are not in any u-fan in Φ then
            Flip the alternating paths P_u, P_v and P_w
            Remove f and any damaged u-fans from U
            Add the u-fan (u, v, w, α', β') to U and Φ
return (α, β)
```

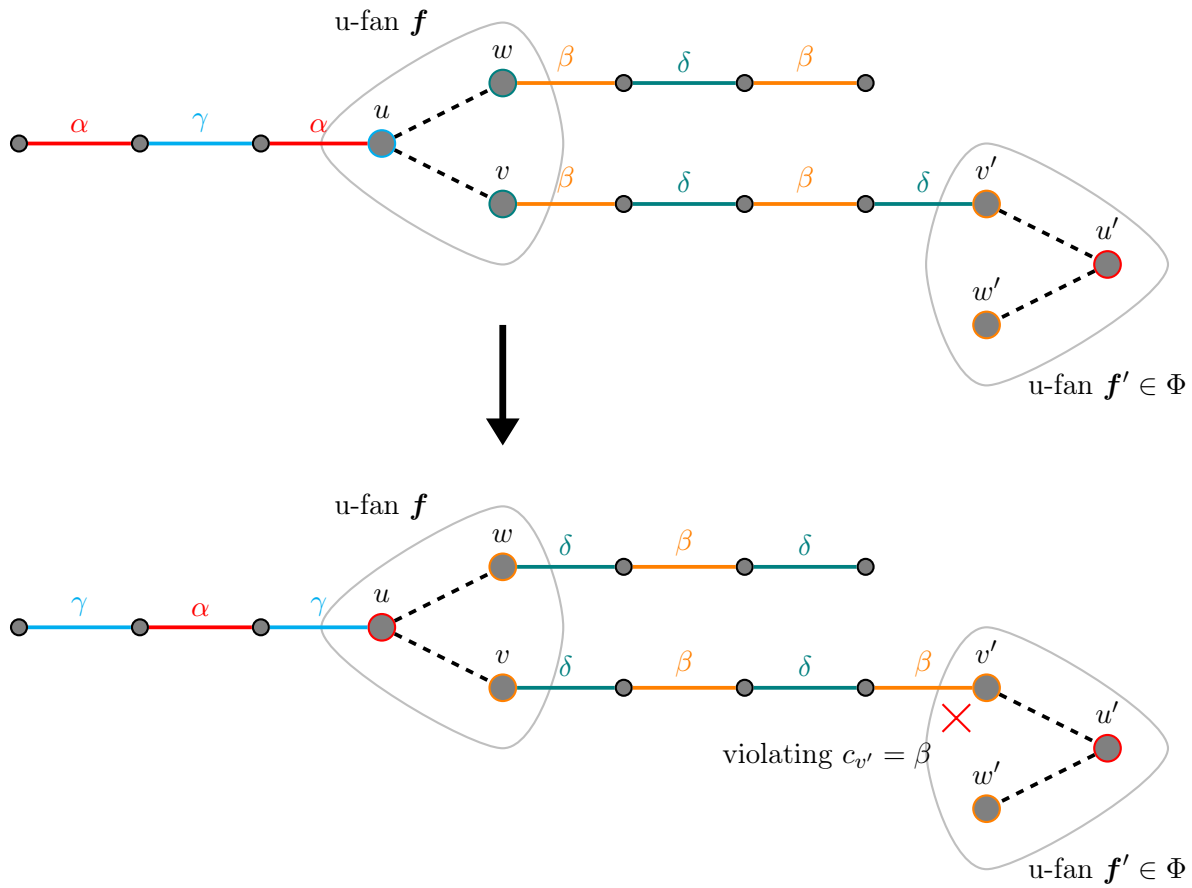

Figure 7.5: In this picture, α is red, β is orange, γ is blue, and δ is green. $(u', v', w') \in \Phi$ is an existing $\{\alpha, \beta\}$-primed u-fan in Φ. If we flip the $\{\alpha, \gamma\}$-alternating path at u and the $\{\beta, \delta\}$-alternating paths at v and w, then we would destroy the property that $\beta \in \mathsf{miss}_\chi(v')$.

Analysis of Prime-U-Fans

We say that an iteration of (the **while** loop in) Prime-U-Fans is *successful* if it adds a u-fan to the set Φ. Otherwise, we say that the iteration *fails* (see Figure 7.5 for an illustration). Note that the algorithm repeatedly performs iterations until it has performed at least $\lambda/(48\Delta)$ successful iterations. The following lemmas summarize the main properties of the subroutine Prime-U-Fans.

Claim 7.4.1. *The total number of edges with color α or β is $O(m/\Delta)$ at any point throughout the run of* Prime-U-Fans.

Proof. Since α and β are initially the two least common colors, there are initially $O(m/\Delta)$ such edges. Since each successful iteration increases the number of such edges by $O(1)$ and there are at most $O(\lambda/\Delta) \leq O(m/\Delta)$ such iterations, the claim follows. □

Claim 7.4.2. *$|\mathcal{U}| \geq (1 - 1/(2\Delta))\lambda$ at any point throughout the run of* Prime-U-Fans.

Proof. Consider a successful iteration of the algorithm where we sample a u-fan $\boldsymbol{f} \in \mathcal{U}$. The algorithm removes $\boldsymbol{f}$ from $\mathcal{U}$, along with any other damaged u-fans in $\mathcal{U}$. It follows from Lemma 7.1.4

that flipping the colors of the alternating paths P_u, P_v and P_w damages at most 6 u-fans in $\mathcal{U}$. Since our algorithm runs for at most $\lambda/(48\Delta)$ iterations, it follows that $|\mathcal{U}| \geq \lambda - 7 \cdot \lambda/(48\Delta)$. □

Lemma 7.4.3. *The u-fans in Φ are all vertex-disjoint and $\{\alpha, \beta\}$-primed at any point throughout the run of* Prime-U-Fans.

Proof. If the algorithm samples a u-fan $\boldsymbol{f}$ that shares a vertex x with a u-fan $\boldsymbol{f}' \in \Phi$, then the iteration fails. Thus, the u-fans in Φ are all vertex-disjoint.

Now, note that whenever we add a u-fan $(u, v, w, \alpha', \beta')$ to Φ, we do this immediately after flipping the paths P_u, P_v, and P_w. Thus, we have that $\alpha' \in \mathsf{miss}_\chi(u)$ and $\beta' \in \mathsf{miss}_\chi(v) \cap \mathsf{miss}_\chi(w)$, so the u-fan is $\{\alpha, \beta\}$-primed when we add it to Φ. Since we only flip the paths in Line 13 if their endpoints do not touch any u-fans in Φ, this operation cannot change what colors are available at vertices in u-fans in Φ, and hence cannot change whether or not any u-fan in Φ is $\{\alpha, \beta\}$-primed. □

The following standard claim bounds the total length of all maximal $\{c, \cdot\}$-alternating paths.

Claim 7.4.4. *For any color $c \in [\Delta + 1]$, the total length of all maximal $\{c, \cdot\}$-alternating paths in χ is at most $4m$.*

Proof. Let $\mathcal{P}_c$ denote the set of all such alternating paths. First note that the total length of all alternating paths in $\mathcal{P}_c$ with length 1 is at most m. Now, let $c' \in [\Delta + 1] \setminus \{c\}$ and let P be a maximal $\{c, c'\}$-alternating path with $|P| \geq 2$. We can observe that at least a third of the edges in P have color c', and that each edge with color c' only appears in one path in $\mathcal{P}_c$. Thus, we have that

$$\sum_{P \in \mathcal{P}_c} |P| \leq m + 3 \cdot |\{e \in E \mid \chi(e) \neq c\}| \leq 4m.$$ □

Lemma 7.4.5. *Each iteration of* Prime-U-Fans *succeeds with probability at least $1/4$.*

Proof. Consider the state of the algorithm at the start of some iteration. For each u-fan $\boldsymbol{f}' \in \mathcal{U}$, let $P_x(\boldsymbol{f}')$ denote the alternating path starting at $x \in \boldsymbol{f}'$ that is considered by the algorithm if it samples $\boldsymbol{f}' \in \mathcal{U}$. We define the *cost* of the u-fan $\boldsymbol{f}'$ to be

$$\mathtt{cost}(\boldsymbol{f}') := \sum_{x \in \boldsymbol{f}'} |P_x(\boldsymbol{f}')|.$$

Let $\mathcal{U}^\star \subseteq \mathcal{U}$ denote the subset of u-fans $\boldsymbol{f}' \in \mathcal{U}$ such that none of the alternating paths $\{P_x(\boldsymbol{f}')\}_{x \in \boldsymbol{f}'}$ have endpoints at some u-fan in Φ. We can see that the iteration is successful if and only if the u-fan $\boldsymbol{f} \in \mathcal{U}$ sampled during the iteration satisfies $\mathtt{cost}(\boldsymbol{f}) \leq 128m/\lambda$ (see Line 10) and $\boldsymbol{f} \in \mathcal{U}^\star$ (see Line 12). Thus, we now show that this happens with probability at least $1/4$.

We first begin with the following claim.

Claim 7.4.6. *Let P be an $\{\alpha, \cdot\}$- or $\{\beta, \cdot\}$-alternating path in χ. Then there are at most 2 u-fans $\boldsymbol{f}' \in \mathcal{U}$ that, if sampled during the iteration, might cause the algorithm to flip P.*

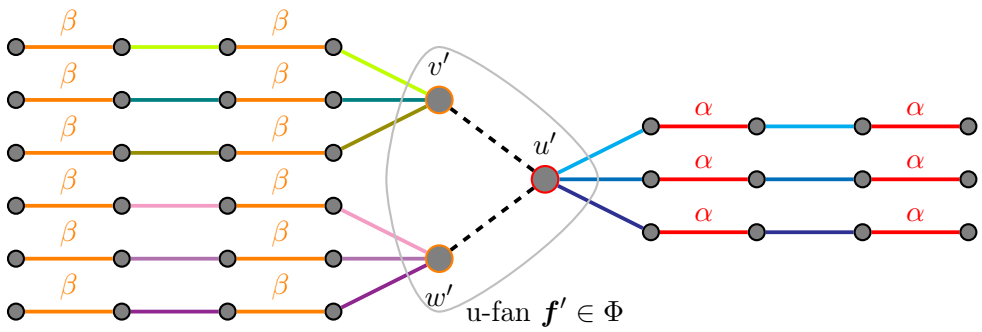

Figure 7.6: In this picture, α is red, β is orange, and $\boldsymbol{f}' = (u', v', w', \alpha, \beta) \in \Phi$ is a u-fan, and we have drawn 9 different $\{\alpha, \cdot\}$- or $\{\beta, \cdot\}$-alternating paths starting from $\boldsymbol{f}'$.

Proof. Suppose that the path P is a $\{c, c'\}$-alternating path for some colors $c \in \{\alpha, \beta\}$ and $c' \notin \{\alpha, \beta\}$ with endpoints x and y.[9] Since the collection of u-fans $\mathcal{U}$ is separable, we know that at most one u-fan in $\mathcal{U}$ containing x (resp. y) has c' assigned as the color missing at x (resp. y). Thus, we must sample one of these 2 u-fans for the algorithm to flip the path P. □

Let $\mathcal{P}$ denote the set of all maximal $\{\alpha, \cdot\}$- and $\{\beta, \cdot\}$-alternating paths in χ. We can see that the expected cost of the u-fan $\boldsymbol{f}$ sampled during the iteration is

$$\mathbb{E}[\texttt{cost}(\boldsymbol{f})] = \frac{1}{|\mathcal{U}|} \cdot \sum_{\boldsymbol{f} \in \mathcal{U}} \sum_{x \in \boldsymbol{f}} |P_x(\boldsymbol{f})| \leq \frac{2}{|\mathcal{U}|} \cdot \sum_{P \in \mathcal{P}} |P| \leq \frac{32m}{\lambda},$$

where we are using the facts that (1) each path in $P \in \mathcal{P}$ appears at most twice while summing over the paths $|P_x(\boldsymbol{f})|$ by Claim 7.4.6, (2) that $|\mathcal{U}| \geq \lambda/2$ by Claim 7.4.2, and (3) that that the total length of all paths in $\mathcal{P}$ is $8m$ by Claim 7.4.4. Applying Markov's inequality, it follows that

$$\Pr\left[\texttt{cost}(\boldsymbol{f}) \geq \frac{128m}{\lambda}\right] \leq \frac{1}{4}.$$

Since each u-fan in Φ has 3 vertices and at most 2Δ $\{\alpha, \cdot\}$- or $\{\beta, \cdot\}$-alternating paths end at each of these vertices, we know that there are at most $6\Delta|\Phi|$ alternating paths that could be considered during some iteration that end at a u-fan in Φ. By Claim 7.4.6, there are at most $12\Delta|\Phi|$ u-fans in $\mathcal{U}$ that could cause the algorithm to consider one of these paths (see Figure 7.6 for an illustration). Thus, at least $|\mathcal{U}| - 12\Delta|\Phi|$ of the u-fans in $\mathcal{U}$ are contained in $\mathcal{U}^\star$. It follows that

$$|\mathcal{U}^\star| \geq |\mathcal{U}| - 12\Delta|\Phi| \geq |\mathcal{U}| - \frac{\lambda}{4} \geq \frac{|\mathcal{U}|}{2},$$

where we are using the facts that $|\Phi| \leq \lambda/(48\Delta)$ and $|\mathcal{U}| \geq \lambda/2$ from Claim 7.4.2. Since $\boldsymbol{f}$ is sampled uniformly at random from $\mathcal{U}$, it follows that $\Pr[\boldsymbol{f} \in \mathcal{U}^\star] \geq 1/2$. The lemma follows by applying a union bound.

[9] Note that the algorithm never flips $\{\alpha, \beta\}$-alternating paths.

□

Lemma 7.4.7. *Each iteration of* Prime-U-Fans *takes time* $O(m/\lambda)$.

Proof. Using standard data structures, each iteration can be implemented in time proportional to the length of the alternating paths P_u, P_v and P_w considered by the algorithm during the iteration. We can check if $|P_u|+|P_v|+|P_w| \leq 128m/\lambda$ in $O(m/\lambda)$ time by traversing these paths and aborting if we notice that their total length exceeds $128m/\lambda$. If their total length is at most $128m/\lambda$, then we can flip these paths and update the collection $\mathcal{U}$ in $O(m/\lambda)$ time. □

Lemma 7.4.8. *The subroutine* Prime-U-Fans *runs in time* $O(m \log n/\Delta)$ *with high probability.*

Proof. Since each iteration of Prime-U-Fans succeeds with probability at least $1/4$ by Lemma 7.4.5 and the subroutine performs iterations until it succeeds $\lambda/(48\Delta)$ times, it follows that it performs at most $O(\lambda \log n/\Delta)$ iterations with high probability. Since each iteration takes $O(m/\lambda)$ time by Lemma 7.4.7, it follows that the total running time is $O(m \log n/\Delta)$ with high probability. □

7.4.2 The Subroutine Activate-U-Fans

As input, this subroutine is given a graph G, a partial $(\Delta+1)$-edge coloring χ of G, and a subset $\Phi \subseteq \mathcal{U}$ of μ vertex-disjoint $\{\alpha,\beta\}$-primed u-fans such that at most $O(m/\Delta)$ edges have color α or β. The subroutine repeatedly picks any $\boldsymbol{f} \in \Phi$ and proceeds to activate the u-fan. It repeats this process until $\Phi = \emptyset$. The pseudocode in Algorithm 11 gives a formal description of the subroutine.

Algorithm 11: Activate-U-Fans$(\mathcal{U}, \alpha, \beta)$

```
while Φ ≠ ∅ do
    Let f ∈ Φ
    Let P and P′ be the {α, β}-alternating paths starting at the leaves of f
    Activate the u-fan f by flipping the path P or P′
    Remove f and any damaged u-fans from U and Φ
```

Analysis of Activate-U-Fans

The following lemmas summarise the key properties of the subroutine Activate-U-Fans.

Lemma 7.4.9. *The subroutine* Activate-U-Fans *extends the coloring* χ *to at least* $\mu/2$ *more edges.*

Proof. During each iteration of the **while** loop, we activate a u-fan $\boldsymbol{f}$ and extend the coloring to an uncolored edge in $\boldsymbol{f}$. By Lemma 7.1.4, this process damages at most 2 u-fans in $\mathcal{U}$ (including $\boldsymbol{f}$). Thus, in each iteration $|\Phi|$ decreases by at most 2, and hence we extend the coloring to at least $\mu/2$ more edges across all the iterations performed by the subroutine. □

Lemma 7.4.10. *The subroutine* Activate-U-Fans *has a running time of* $O(m/\Delta)$.

Proof. Let $\mathcal{P}$ denote the collection of maximal $\{\alpha, \beta\}$-alternating paths in the coloring χ when we first call Activate-U-Fans. Since at most $O(m/\Delta)$ edges have color α and β in this coloring, we have that the total length of the paths in $\mathcal{P}$ is $O(m/\Delta)$ since the paths in $\mathcal{P}$ are all vertex-disjoint.

Throughout the run of Activate-U-Fans, we only modify χ by flipping the colors of $\{\alpha, \beta\}$-alternating paths (see Line 4). Thus, the structure of the paths in $\mathcal{P}$ does not change (but their colors might be flipped). Furthermore, each time the subroutine flips an $\{\alpha, \beta\}$-alternating path, it removes any u-fan from Φ that contains an endpoint of this alternating path (see Line 5). Thus, each path in $\mathcal{P}$ is only flipped at most once.

Using standard data structures, each iteration of the **while** loop can be implemented in time proportional to the length of the alternating path that is flipped during the iteration (see Line 4). It follows that the total running time of the subroutine across all iterations is at most $\sum_{P \in \mathcal{P}} O(|P|) \leq O(m/\Delta)$. □

Claim 7.4.11. *$|\mathcal{U}|$ decreases by at most 2μ throughout the run of* Activate-U-Fans.

Proof. During each iteration, we flip an alternating path and remove the damaged u-fans from $\mathcal{U}$ and Φ. It follows from Lemma 7.1.4 that each iteration removes at most 2 u-fans from these sets. Since we perform at most μ iterations, the claim follows. □

7.4.3 Analysis of Color-U-Fans: Proof of Lemma 7.2.3

Given a separable collection of λ u-fans $\mathcal{U}$, the algorithm Color-U-Fans repeatedly calls Prime-U-Fans and Activate-U-Fans with the set $\mathcal{U}$ as described in Algorithm 9. It repeats this process for $\Delta/2$ iterations. It follows from Claims 7.4.2 and 7.4.11 that $|\mathcal{U}|$ decreases by at most a $(1 - 1/\Delta)$ factor during each iteration. Thus, by Bernoulli's inequality, we get that

$$|\mathcal{U}| \geq \lambda \cdot \left(1 - \frac{1}{\Delta}\right)^{\Delta/2} \geq \frac{\lambda}{2}$$

throughout the entire run of the algorithm. In each iteration of Algorithm 9, we extend the coloring to $\Omega(\lambda/\Delta)$ edges in $O(m \log n/\Delta)$ time w.h.p. Thus, in total, we extend the coloring to $\Omega(\lambda)$ edges in $O(m \log n)$ time w.h.p.

7.5 Implementation and Data Structures

In this section, we describe the key data structures that we use to implement an edge coloring χ and a separable collection $\mathcal{U}$, allowing us to efficiently implement the operations performed by our algorithms. We first describe the data structures and then show how they can be used to efficiently implement the queries described in Section 7.1.2.

Implementing an Edge Coloring. Let $G = (V, E)$ be a graph of maximum degree Δ and let $\mathcal{C} := [\Delta + 1] \cup \{\perp\}$. We implement an edge coloring $\chi : E \longrightarrow \mathcal{C}$ of G using the following:

- The map $\phi : E \longrightarrow \mathcal{C}$ where $\phi(e) := \chi(e)$ for all $e \in E$.
- The map $\phi' : V \times \mathcal{C} \longrightarrow E$ where $\phi'_u(c) := \{e \ni u \mid \chi(e) = c\}$.
- The set $\chi^{-1}(c) := \{e \in E \mid \chi(e) = c\}$, for all $c \in \mathcal{C}$.
- The set $\mathsf{miss}_\chi(u) \cap [\deg_G(u) + 1]$, for all $u \in V$.[10]

We implement all of the maps and sets using hashmaps, allowing us to perform membership queries, insertions and deletions in $O(1)$ time (see Proposition 7.1.6).[11] The map ϕ' allows us to check if a color $c \in [\Delta + 1]$ is available at a vertex $u \in V$ in $O(1)$ time, and if it is not, to find the edge $e \ni u$ with $\chi(e) = c$. The sets $\{\chi^{-1}(c)\}_{c \in \mathcal{C}}$ allow us to easily return all edges with a specific color (including $\perp$). Furthermore, we can determine which color classes are the least common in $O(1)$ time.[12] Each time an edge e changes color under χ, we can easily update all of these data structures in $O(1)$ time. Furthermore, given $O(1)$ time query access to an edge coloring χ, we can initialize these data structures in $O(m)$ time. We note that χ is a proper edge coloring if and only if $|\phi'_u(c)| \leq 1$ for all $u \in V, c \in [\Delta + 1]$.

Since the hashmap used to implement ϕ stores m elements, it follows that it can be implemented with $O(m)$ space. Similarly, the map ϕ' stores $2m$ elements (if $\{e \ni u \mid \chi(e) = c\} = \emptyset$, then we do not store anything for $\phi'_u(c)$) and thus can be implemented with $O(m)$ space since each element has size $O(1)$ (recall that $|\phi'_u(c)| \leq 1$ since the coloring is proper). Since each set $\chi^{-1}(c)$ can be stored in space $O(|\chi^{-1}(c)|)$ and $\sum_c |\chi^{-1}(c)| = m$, these sets can be implemented in space $O(m)$. Similarly, since $\sum_u |\mathsf{miss}_\chi(u) \cap [\deg_G(u) + 1]| = O(m)$, the sets $\mathsf{miss}_\chi(u) \cap [\deg_G(u) + 1]$ can also be implemented in $O(m)$ space.

Implementing a Separable Collection. We implement a separable collection $\mathcal{U}$ in a similar manner using the following:

- The map $\psi : V \times [\Delta + 1] \longrightarrow \mathcal{U}$ where $\psi_u(c) := \{\boldsymbol{g} \in \mathcal{U} \mid u \in \boldsymbol{g}, c_{\boldsymbol{g}}(u) = c\}$.
- The set $C_{\mathcal{U}}(u) := \{c_{\boldsymbol{g}}(u) \mid \boldsymbol{g} \in \mathcal{U}, u \in \boldsymbol{g}\}$, for all $u \in V$.
- The set $\overline{C}_{\mathcal{U}}(u) := (\mathsf{miss}_\chi(u) \cap [\deg_G(u) + 1]) \setminus C_{\mathcal{U}}(u)$, for all $u \in V$.

We again implement all of the maps and sets using hashmaps, allowing us to access and change entries in $O(1)$ time. We note that, since $\mathcal{U}$ is separable, $|\psi_u(c)| \leq 1$ for all $u \in V, c \in [\Delta + 1]$. Thus, we can determine the size of $\mathcal{U}$ and also sample from $\mathcal{U}$ uniformly at random in $O(1)$ time.[13] Each time we remove a color $c \in [\Delta + 1]$ from the palette $\mathsf{miss}_\chi(u)$ of a vertex $u \in V$, we can update $\overline{C}_{\mathcal{U}}(u)$ in $O(1)$ time and check $\psi_u(c)$ in $O(1)$ time to find any u-component that has been

[10] We take this intersection with $[\deg_G(u) + 1]$ instead of maintaining $\mathsf{miss}_\chi(u)$ directly to ensure that the space complexity and initialization time of the data structures are $\tilde{O}(m)$ and not $\Omega(\Delta n)$.

[11] By using balanced search trees instead of hashmaps, we can make the data structures deterministic while increasing the time taken to perform these operations to $O(\log n)$.

[12] For example, we can then maintain a list of colors $c \in \mathcal{C}$ sorted by the values of $|\chi^{-1}(c)|$.

[13] For example, we can sample a number $r \sim [|\mathcal{U}|]$ u.a.r. and then return the r^{th} element in the hashmap that implements ψ.

damaged. Each time we add or remove a u-component from $\mathcal{U}$, we can update all of these data structures in $O(1)$ time. Furthermore, we can initialize these data structures for an empty collection in $O(m)$ time by creating an empty map ψ, empty sets $C_{\mathcal{U}}(u)$ for each $u \in V$ and copying the sets $\overline{C}_{\mathcal{U}}(u) = \mathsf{miss}_{\chi}(u) \cap [\deg_G(u) + 1]$ for each $u \in V$ which are maintained by the data structures for the edge coloring χ. Since $\mathcal{U}$ is separable, we can see that $\overline{C}_{\mathcal{U}}(u) \neq \emptyset$. Thus, whenever we want a color from the set $\mathsf{miss}_{\chi}(u) \setminus C_{\mathcal{U}}(u)$, it suffices to take an arbitrary color from $\overline{C}_{\mathcal{U}}(u)$.

For each $\boldsymbol{g} \in \mathcal{U}$, we can see that $\boldsymbol{g}$ is contained at most 3 times in ψ. Thus, the total space required to store the hashmap that implements ψ is $O(|\mathcal{U}|)$. Since $\mathcal{U}$ is separable, the u-components in $\mathcal{U}$ are edge-disjoint, and thus $|\mathcal{U}| \leq m$. It follows that the map ψ can be stored with $O(m)$ space. For each $u \in V$, we can observe that $|C_{\mathcal{U}}(u)| \leq \deg_G(u)$ since at most $\deg_G(u)$ many u-components in $\mathcal{U}$ contain the vertex u, and $|\overline{C}_{\mathcal{U}}(u)| \leq \deg_G(u) + 1$ since $\overline{C}_{\mathcal{U}}(u) \subseteq [\deg_G(u) + 1]$. Thus, the total space required to store the sets $\{C_{\mathcal{U}}(u)\}_{u \in V}$ and $\{\overline{C}_{\mathcal{U}}(u)\}_{u \in V}$ is $O(m)$.

7.5.1 Implementing the Operations from Section 7.1.2

We now describe how to implement each of the operations from Section 7.1.2.

Implementing Initialize(G, χ): Suppose that we are given the graph G and $O(1)$ time query access to an edge coloring χ of G. We can initialize the data structures used to maintain the maps ϕ and ϕ' in $O(m)$ time. We can then scan through the edges $e \in E$ and initialize the sets $\chi^{-1}(c)$ in $O(m)$ time. Finally, we can scan through the vertices $u \in V$ and initialize the sets $\mathsf{miss}_{\chi}(u) \cap [\deg_G(u) + 1]$ in $O(m)$ time. We can then initialize the data structures for an empty separable collection in $O(m)$ time by creating an empty map ψ and, for each $u \in V$, initializing the sets $C_{\mathcal{U}}(u) \leftarrow \emptyset$ and $\overline{C}_{\mathcal{U}}(u) \leftarrow \mathsf{miss}_{\chi}(u) \cap [\deg_G(u) + 1]$.

Implementing Insert$_{\mathcal{U}}(\boldsymbol{g})$: By performing at most 3 queries to the map ψ, we can check if $\mathcal{U} \cup \{\boldsymbol{g}\}$ is separable. If so we can update ψ and the sets $C_{\mathcal{U}}(x)$ and $\overline{C}_{\mathcal{U}}(x)$ for $x \in \boldsymbol{g}$ in $O(1)$ time in order to insert $\boldsymbol{g}$ into $\mathcal{U}$. Otherwise, we return `fail`.

Implementing Delete$_{\mathcal{U}}(\boldsymbol{g})$: We can first make a query to ψ to ensure that $\boldsymbol{g} \in \mathcal{U}$. If so, we can update ψ and the sets $C_{\mathcal{U}}(x)$ and $\overline{C}_{\mathcal{U}}(x)$ for $x \in \boldsymbol{g}$ in $O(1)$ time to remove $\boldsymbol{g}$ from $\mathcal{U}$.

Implementing Find-Component$_{\mathcal{U}}(x, c)$: We make a query to ψ by checking if there is an element $\psi_x(c)$. If no such element is contained in ψ, then return `fail`. Otherwise, return the unique u-component in the set $\psi_x(c)$.

Implementing Missing-Color$_{\mathcal{U}}(x)$: Return an arbitrary color from the set $\overline{C}_{\mathcal{U}}(x)$.

7.6 The Final Algorithm: Proof of Theorem 7.0.1

Up until this point in the presentation of our algorithm and its analysis, we made no attempt in optimizing the logarithmic runtime factors, which led to an algorithm with $O(m \log^3 n)$ time for finding a $(\Delta + 1)$-edge coloring with high probability in Theorem 7.2.1. We now show that we can

further optimize the algorithm and obtain an $O(m \log n)$ time algorithm and conclude the proof of Theorem 7.0.1.

Let us first start by listing where the $\log(n)$-terms come from in the proof of Theorem 7.2.1:

1. In Lemma 7.2.3, when coloring u-fans, we are losing an $O(\log n)$ factor to ensure the probabilistic guarantees of the algorithm hold with high probability in each step (see Lemma 7.4.8).

2. Our application of Lemmas 7.2.2 and 7.2.3 can color a constant fraction of remaining uncolored edges, hence, we need to run them $O(\log n)$ times to color all uncolored edges (see Lemma 7.2.4).

3. And finally, the entire framework of reducing $(\Delta+1)$-coloring to extending the coloring to $O(m/\Delta)$ uncolored edges using Eulerian partition technique (in the proof of Theorem 7.2.1) leads to a recursion depth of $O(\log n)$ leading to another $O(\log n)$ overhead in the runtime.

All in all, these factors led to the $O(m \log^3 n)$ bound of our algorithm in Theorem 7.2.1.

We now show how these log factors can be reduced to a single one. The two main ideas are: (1) relaxing the requirement of Lemma 7.2.3 so that its runtime holds in expectation, plus a suitable tail bound. Then, instead of maintaining high probability bound on *each* invocation of this lemma, we only bound the runtime of *all* invocations of this lemma together with high probability; (2) relaxing the Eulerian partition technique to color most of the graph recursively, instead of the entire graph, and then taking care of the remaining uncolored edges at the end.

We start by presenting a more fine-grained version of two of our main technical lemmas in the proof of Theorem 7.2.1 (Lemma 7.2.3 and Lemma 7.2.4; recall that Lemma 7.2.2 already runs in linear time deterministically), which correspond to part (1) above, and then use these to present part (2) of above and conclude the proof.

7.6.1 Fine-Grained Variants of Lemma 7.2.3 and Lemma 7.2.4

We remove the $O(\log n)$-term of Lemma 7.2.3 by focusing on the expected runtime of the algorithm (plus a crucial tail inequality). In the next part, we show how to recover the final result even from this weaker guarantee.

Lemma 7.6.1 (A slight modification of Lemma 7.2.3)**.** *There is an algorithm that, given a graph G, a partial $(\Delta+1)$-edge coloring χ and a separable collection of λ u-fans, extends χ to $\Omega(\lambda)$ in T (randomized) time such that for some $T_0 = O(m)$, we have,*

$$\mathbb{E}[T] \leq T_0 \quad \textit{and for all } \delta > 0 \quad \Pr[T \geq (1+\delta) \cdot 2T_0] \leq \exp\left(-\frac{\delta \cdot \lambda}{100}\right).$$

Proof. The amortized runtime for coloring each single edge in Lemma 7.2.3 is $O(m/\lambda)$ by Lemma 7.4.7 assuming the coloring succeeds, which happens with probability at least $1/4$ by Lemma 7.4.5. We emphasize that this is in an average sense: the algorithm Color-U-Fans first uses Prime-U-Fans to prime $\Omega(\lambda/\Delta)$ u-fans in expected $O(m/\lambda)$ time per u-fan and thus $O((\lambda/\Delta) \cdot (m/\lambda)) = O(m/\Delta)$

expected total time, and then deterministically colors them in $O(m/\Delta)$ time using Activate-U-Fans. It then repeats this process $\Delta/2$ times to color $\Omega(\lambda)$ edges, implying that $\mathbb{E}[T] = O(m)$ time.

We now prove the desired tail inequality on T as well. Let us number all iterations of the while-loop in Prime-U-Fans from 1 to t, across *all* $\Delta/2$ times Prime-U-Fans is called in Color-U-Fans: for the i^{th} iteration, define an indicator random variable X_i which is 1 iff this iteration of the while-loop is successful in priming a u-fan (i.e., increasing the size of Φ). Additionally, for every such i, define $S_i := \sum_{j=1}^{i} X_j$ which denotes the number of successful iterations after the algorithm has done i iterations in total. Thus, t is the smallest integer where $S_t = \gamma \cdot \lambda$ for some integer $\gamma \in (0, 1)$ which is the fraction of u-fans the algorithm colors (precisely, $\gamma = 1/48 \cdot 1/2 \geq 1/100$).

For any $i \geq 1$, S_i stochastically dominates the binomial distribution with parameters i and $p = 1/4$ (by Lemma 7.4.5 each iteration is successful with probability at least $1/4$). Hence, for $\delta > 0$, by concentration results for the binomial distribution,

$$\Pr[t \geq (1+\delta) \cdot 8\gamma \cdot \lambda] \leq \exp\left(-\frac{(2(1+\delta))^2}{2+2(1+\delta)} \cdot 4\gamma \cdot \lambda\right) \leq \exp\left(-\frac{\delta \cdot \lambda}{100}\right),$$

using a loose upper bound in the last step. Given that the runtime of the algorithm is $t \cdot O(m/\lambda)$, the tail inequality follows. □

We note Lemma 7.6.1 guarantees that as long as $\lambda = \omega(\log n)$, the $O(m)$ runtime of the algorithm also holds with high probability. For smaller values of λ (which happens only as a corner case in the algorithm[14]), we need the specific tail inequality proven in the lemma instead.

Furthermore, we provide a similarly fine-grained version of Lemma 7.2.4 that will be used in the last step of the argument to bypass the $O(\log n)$ factor loss of the original lemma.

Lemma 7.6.2 (A slight modification of Lemma 7.2.4)**.** *There is an algorithm that given a graph G, a partial $(\Delta+1)$-edge coloring χ of G with λ uncolored edges, and an integer $\lambda_0 \leq \lambda$, extends χ to all but λ_0 uncolored edges in T (randomized) time such that for some $T_0 = O(m \cdot \log(\lambda/\lambda_0) + \Delta\lambda)$, we have,*

$$\mathbb{E}[T] \leq T_0 \quad \textit{and for all } \delta > 0 \quad \Pr[T \geq (1+\delta) \cdot 2T_0] \leq \exp\left(-\frac{\delta \cdot \lambda_0}{200}\right).$$

Proof. The algorithm is verbatim as in Lemma 7.2.4 by replacing Lemma 7.2.3 with Lemma 7.6.1: repeatedly color a constant fraction of remaining edges as long as λ_0 uncolored edges remain. Since we start with λ uncolored edges and finish with λ_0 many, and reduce uncolored edges by a constant factor each time, we apply Lemma 7.2.2 and Lemma 7.6.1 for a total of $O(\log(\lambda/\lambda_0))$ times. Moreover, the $\Delta\lambda$ terms in the runtime of Lemma 7.2.2 form a geometric series and thus their contribution to the runtime is $O(\Delta\lambda)$ in total. This proves the expected bound on the runtime, i.e., $\mathbb{E}[T] \leq T_0 = O(m \cdot \log(\lambda/\lambda_0) + \Delta\lambda)$.

As for the tail bound, in each application of Lemma 7.6.1 with an intermediate value $\lambda' \in [\lambda_0, \lambda]$,

[14] The only case in our algorithm where this tail bound cannot be replaced with a high probability bound is when $m/\Delta = o(\log n)$ and at the same time $\Delta \log^2 \Delta = \omega(m \log \Delta)$, which means $m = o(n \log n)$ and yet $\Delta = \omega(n/\log n)$.

and $T_0' = O(m)$ being the T_0-parameter of Lemma 7.6.1, we have,

$$\Pr\left[\text{runtime of Lemma 7.6.1 with } \lambda' \text{ uncolored edges} \geq (1+\delta)\cdot 2T_0'\right] \leq \exp\left(-\frac{\delta\cdot\lambda'}{100}\right).$$

Thus, by union bound,

$$\begin{aligned}\Pr\left[\text{total runtime} \geq k\cdot T_0\right] &\leq \sum_{\lambda'} \exp\left(-\frac{\delta\cdot\lambda'}{100}\right)\\ &\leq \sum_{i=0}^{\infty}\cdot\exp\left(-\frac{\delta\cdot\lambda_0\cdot\gamma^i}{100}\right)\\ &\text{(as number of uncolored edges drops by some constant factor, say, } \gamma = 1+\Theta(1) \text{ each time)}\\ &\leq \exp\left(-\frac{\delta\cdot\lambda_0}{200}\right),\end{aligned}$$

by (loosely) upper bounding the sum of the geometric series using its first term. □

Before moving on from this subsection, we mention the following specialized concentration inequality that we need in order to be able to exploit the tail bounds proven in Lemma 7.6.1 and Lemma 7.6.2. The proof follows a standard moment generating function argument and is postponed to Section 7.10.

Proposition 7.6.3. *Let $\{X_i\}_{i=1}^n$ be n independent non-negative random variables associated with parameters $\{\alpha_i\}_{i=1}^n$ and $\{\beta_i\}_{i=1}^n$ such that for each $i\in[n]$, $\alpha_i,\beta_i\geq 1$, and for every $\delta>0$,*

$$\Pr\left[X_i \geq (1+\delta)\cdot\alpha_i\right] \leq \exp\left(-\delta\cdot\beta_i\right);$$

then, for every $t\geq 0$,

$$\Pr\left[\sum_{i=1}^n X_i \geq \sum_{i=1}^n \alpha_i + t\right] \leq \exp\left(-\left(\min_{i=1}^n \frac{\beta_i}{2\alpha_i}\right)\cdot\left(t - 2\sum_{i=1}^n\frac{\alpha_i}{\beta_i}\right)\right).$$

7.6.2 Proof of Theorem 7.0.1

We are now ready to use the more fine-grained versions of our main technical lemmas to $(\Delta+1)$ edge color the graph in $O(m\log n)$ randomized time, and conclude the proof of Theorem 7.0.1.

The first part is based on a modification to the Eulerian partition approach used in the proof of Theorem 7.2.1. Notice that the runtime of this algorithm is even $O(m\log\Delta)$ and not $O(m\log n)$ (the distinction at this point is irrelevant for us in proving Theorem 7.0.1, but we state the result this way so we can use it later in Section 7.8 as well).

Lemma 7.6.4. *There is an algorithm that given a graph G, finds a $(\Delta+1)$-edge coloring of all but (exactly) m/Δ edges in $O(m\log\Delta)$ time with high probability.*

Proof. Consider the following recursive algorithm. Find an Eulerian tour of G and partition G into two edge-disjoint subgraphs G_1 and G_2 on the same vertex set such that $\Delta(G_1), \Delta(G_2) \leq \lceil \Delta/2 \rceil$ and $m(G_1), m(G_2) \leq \lceil m/2 \rceil$. For $i \in \{1, 2\}$, recursively, find a $(\Delta(G_i)+1)$-edge coloring χ_i of G_i that leaves $m(G_i)/\Delta(G_i) = O(m/\Delta)$ edges uncolored. Combining χ_1 and χ_2 and uncoloring the two smallest color classes, gives a $(\Delta+1)$-coloring χ of all but $\lambda := O(m/\Delta)$ edges. We then run Lemma 7.6.2 to reduce the number of uncolored edges to $\lambda_0 := m/\Delta$ edges. The correctness of the algorithm thus follows immediately.

For the runtime, we have $k := O(\Delta)$ sub-problems in total, with 2^d subproblems on $\lceil m/2^d \rceil$ edges and maximum degree $\lceil \Delta/2^d \rceil$ for $d \leq \lceil \log \Delta \rceil$. Let $X_1, \ldots, X_k$ denote the runtime of these subproblems and thus $X := \sum_{i=1}^{k} X_i$ is the total runtime. We have,

$$\mathbb{E}[X] = \sum_i \mathbb{E}[X_i] = \sum_{d=1}^{\lceil \log \Delta \rceil} 2^d \cdot O\left(\frac{m}{2^d} \cdot \log(\lambda/\lambda_0) + \frac{\Delta}{2^d} \cdot \lambda\right) = O(m \log \Delta)$$

by Lemma 7.6.2, since $\lambda = O(m/\Delta)$, $\lambda_0 = m/\Delta$, and thus $\log(\lambda/\lambda_0) = O(1)$ and $\Delta \cdot \lambda = O(m)$. Moreover, by Lemma 7.6.2, for the sub-problem corresponding to X_i at some level d of the recursion, $T_0^d = O(m/2^d)$, and every $\delta \geq 0$,

$$\Pr\left[X_i \geq (1+\delta) \cdot 2T_0^d\right] \leq \exp\left(-\delta \cdot \frac{\lambda_0}{200}\right) = \exp\left(-\delta \cdot \frac{m}{200\Delta}\right).$$

For this X_i, define $\alpha_i := 2T_0^d$ and $\beta_i := m/200\Delta$. We can now apply Proposition 7.6.3 and for

$$t := 1000 \sum_{i=1}^{k} \alpha_i = 1000 \cdot \sum_{d=1}^{\lceil \log \Delta \rceil} 2^d \cdot 2T_0^d = O(m \log \Delta),$$

have,

$$\begin{aligned}
\Pr[X \geq t] &\leq \exp\left(-\left(\min_{i=1}^{n} \frac{\beta_i}{2\alpha_i}\right) \cdot \left(t - 2\sum_{i=1}^{n} \frac{\alpha_i}{\beta_i}\right)\right) \\
&= \exp\left(-\frac{\Theta(1)}{\Delta} \cdot \left(t - 2 \cdot \frac{t \cdot 200\Delta}{1000 \cdot m}\right)\right) \\
&\leq \exp\left(-\frac{\Theta(1)}{\Delta} \cdot (3t/5)\right) && \text{(as } \Delta \leq m \text{ always trivially)} \\
&= \exp\left(-\Theta(1) \cdot \frac{m \log \Delta}{\Delta}\right),
\end{aligned}$$

where we can make the constant in $\Theta(1)$ arbitrarily large without changing the asymptotic runtime of the algorithm. This probability is always at most $1/\operatorname{poly}(n)$ because either $\Delta < \sqrt{m}$, in which case, the probability is at most $1/2^{\Theta(\sqrt{m})}$, or $\Delta > \sqrt{m}$, and thus $\log \Delta = \Theta(\log n)$ and the probability is $1/\operatorname{poly}(n)$ at the very least. This concludes the proof. □

To conclude the proof, we have the following lemma that colors the remaining m/Δ edges left un-

colored by the algorithm of Lemma 7.6.4. The proof is a straightforward application of Lemma 7.6.2, plus the original Vizing's Fan and Chain approach stated in Lemma 3.2.2.

Lemma 7.6.5. *There is an algorithm that, given a graph G, a partial $(\Delta+1)$-edge coloring χ with $\lambda = m/\Delta$ uncolored edges, extends the coloring to all edges in $O(m \log n)$ time with high probability.*

Proof. We first run our Lemma 7.6.2 with the given parameters $\lambda = m/\Delta$ and $\lambda_0 = 100 \log n$. By Lemma 7.6.2, the expected runtime will be some $T_0 = O(m \log(\lambda/\lambda_0) + \Delta\lambda) = O(m \log n)$ since $\lambda \leq m$ and $\lambda_0 \geq 1$. Moreover, by the same lemma,

$$\Pr[\text{runtime of the algorithm} \geq 2000 \cdot T_0] \leq \exp(-10\lambda_0) \leq \exp(-1000 \log n) = 1/\operatorname{poly}(n),$$

since $\lambda_0 \geq 100 \log n$. Thus, after running this part, in $O(m \log n)$ time, with high probability, we will be left with only $O(\log n)$ uncolored edges. We can then color each remaining uncolored edge in $O(n)$ time using the original Vizing's Fans and Chains approach of Lemma 3.2.2, thus obtaining a $(\Delta+1)$-coloring of the entire graph in $O(m \log n)$ time with high probability. □

Theorem 7.0.1 now follows immediately from Lemma 7.6.4 and Lemma 7.6.5.

7.7 Vizing's Theorem for Multigraphs in Near-Linear Time

We now show how to extend our arguments to *multigraphs*. A generalization of Vizing's theorem to multigraphs shows that any multigraph G with maximum degree Δ and maximum multiplicity μ can be edge colored with $\Delta + \mu$ colors [Viz64] (and this is worst-case optimal for $\mu \leq \Delta/2$ in the sense that not every multigraph admits an edge coloring with fewer colors). We prove the following theorem, which shows how to compute such a coloring in near-linear time.

Theorem 7.7.1. *There is a randomized algorithm that, given an undirected multigraph $G = (V, E)$ on n vertices and m edges with maximum degree Δ and maximum multiplicity μ, finds a $(\Delta+\mu)$-coloring of G in $O(m \log n)$ time with high probability.*

Similar to our results for $(\Delta+1)$-coloring, we first focus on only obtaining an $\tilde{O}(m)$ time algorithm, and then use almost exactly the same argument we used to extend Theorem 7.2.1 to Theorem 7.0.1, to optimize this runtime to $O(m \log n)$ time.

Notation for Multigraphs. We refer to edges in G with the same endpoints as *parallel*. Whenever we refer to an edge e of the multigraph G, we are referring to a specific edge and say that this edge is distinct from its parallel edges.

7.7.1 Vizing Fans and Separable Collection in Multigraphs

Let $G = (V, E)$ be an undirected multigraph on n vertices and m edges with maximum degree Δ and maximum multiplicity μ, and χ be a partial $(\Delta+\mu)$-edge coloring of G. We now describe how to generalize the objects used by our algorithm for simple graphs to deal with multigraphs. We

begin by defining a generalization of Vizing fans for multigraphs [Viz64]. For convenience, we still refer to them as Vizing fans.

Definition 7.7.2 (Vizing fan for multigraphs)**.** *A* Vizing fan *for a multigraph is a sequence* $\mathbf{F} = (u, \alpha), (v_1, e_1, C_1), \ldots, (v_k, e_k, C_k)$ *where* $u, v_1, \ldots, v_k$ *are distinct vertices,* $C_1, \ldots, C_k \subseteq [\Delta + \mu]$ *are subsets of colors and* $e_1, \ldots, e_k$ *are edges such that*

1. $\alpha \in \mathsf{miss}_\chi(u)$ *and* $C_i \subseteq \mathsf{miss}_\chi(v_i)$ *of size* μ *for all* $i \in [k]$*.*
2. $v_1, \ldots, v_k$ *are distinct neighbours of* u *and* e_i *is an edge with endpoints* u *and* v_i *for all* $i \in [k]$*.*
3. $\chi(e_1) = \perp$ *and, for all* $i > 1$*, there exists* $p(i) \in [i-1]$ *such that* $\chi(e_i) \in C_{p(i)}$*.*
4. *The sets* $C_1, \ldots, C_{k-1}, \mathsf{miss}_\chi(u)$ *are mutually disjoint.*
5. *Either* $C_k \cap \mathsf{miss}_\chi(u) \neq \emptyset$ *or* $C_k \cap (C_1 \cup \cdots \cup C_{k-1}) \neq \emptyset$*.*

We say that the Vizing fan $\mathbf{F} = (u, \alpha), (v_1, e_1, C_1), \ldots, (v_k, e_k, C_k)$ is *α-primed*, has *center* u and *leaves* $v_1, \ldots, v_k$. We refer to C_i as the colors of v_i within $\mathbf{F}$. A crucial property is that we can *rotate* colors around the Vizing fan $\mathbf{F}$: given any $i \in [k]$, there is a sequence $1 = i_1 < \cdots < i_\ell = i$ (where $i_{\ell-j} = p^{(j)}(i)$) such that we can set $\chi(e_{i_1}) \leftarrow \chi(e_{i_2}), \ldots, \chi(e_{i_{\ell-1}}) \leftarrow \chi(e_{i_\ell}), \chi(e_{i_\ell}) \leftarrow \perp$. We say that $\mathbf{F}$ is a *trivial* Vizing fan if $C_k \cap \mathsf{miss}_\chi(u) \neq \emptyset$. Note that, if $\mathbf{F}$ is trivial, we can immediately extend the coloring χ to (e_1) by rotating colors around $\mathbf{F}$ to leave (e_1) uncolored and setting $\chi(e_k)$ to be any color in $C_k \cap \mathsf{miss}_\chi(u)$.

For completeness, we prove the following lemma, which shows how to efficiently construct such a Vizing fan.

Lemma 7.7.3 ([Viz64])**.** *Given an uncolored edge* $e = (u, v)$ *and a color* $\alpha \in \mathsf{miss}_\chi(u)$*, we can construct an* α*-primed Vizing fan* $\mathbf{F}$ *with center* u *in* $O(\Delta)$ *time.*

Proof. Let $v_1 = v$, $e_1 = e$ and $C_1 \subseteq \mathsf{miss}_\chi(v)$ be a subset of size μ. If $C_1 \cap \mathsf{miss}_\chi(u) \neq \emptyset$, then we can return $(u, \alpha), (v_1, e_1, C_1)$ as a Vizing fan. Otherwise, we assume inductively that we have constructed a sequence $(u, \alpha), (v_1, e_1, C_1), \ldots, (v_i, e_i, C_i)$ satisfying Conditions 1 to 4 of Definition 7.7.2. In this case, we have that $|C_1 \cup \cdots \cup C_i| = i\mu$. Since there are at most $i\mu - 1$ many colored edges that have u as one endpoint and one of $v_1, \ldots, v_i$ as the other, it follows that there must be some edge $e_{i+1} = (u, v_{i+1})$ such that $\chi(e_{i+1}) \in C_1 \cup \cdots \cup C_i$ and $v_{i+1} \notin \{v_1, \ldots, v_i\}$. Let $C_{i+1} \subseteq \mathsf{miss}_\chi(v_{i+1})$ be a subset of size μ. If either $C_{i+1} \cap \mathsf{miss}_\chi(u) \neq \emptyset$ or $C_{i+1} \cap (C_1 \cup \cdots \cup C_i) \neq \emptyset$, then $(u, \alpha), (v_1, e_1, C_1), \ldots, (v_{i+1}, e_{i+1}, C_{i+1})$ is a Vizing fan and we are done. Otherwise, we continue the induction on this longer sequence. Since this process must terminate within at most Δ/μ steps, this process always returns a Vizing fan. Furthermore, using standard data structures, we can implement this process in $O(\Delta)$ time. □

Vizing Chains in Multigraphs. Let $\mathbf{F} = (u, \alpha), (v_1, e_1, C_1), \ldots, (v_k, e_k, C_k)$ be a non-trivial α-primed Vizing fan, let c be a color in $C_k \cap (C_1, \ldots, C_{k-1})$ and let P denote the maximal $\{\alpha, c\}$-alternating path starting at u. Then, by a similar argument to the case of simple graphs, we

can extend the coloring χ to the edge e_1 by flipping the path P and rotating colors around $\mathbf{F}$. Furthermore, this can be done in time $O(\Delta + |P|)$. We denote a call to the algorithm that extends the coloring in the manner by $\mathsf{Vizing}(\mathbf{F})$.

Separable Collections in Multigraphs. We define a separable collection of u-components $\mathcal{U}$ in a multigraph in the exact same way as simple graphs. We emphasise that, given a separable collection $\mathcal{U}$, any distinct u-components $\boldsymbol{g}_1, \boldsymbol{g}_2 \in \mathcal{U}$ are edge-disjoint but may contain parallel edges. In other words, it is possible for two distinct parallel uncolored edges e_1 and e_2 to be contained in u-components within the separable collection $\mathcal{U}$.

We define $\mathcal{U}$-avoiding Vizing fans for multigraphs in the following analogous way.

Definition 7.7.4. *Let $\mathcal{U}$ be a separable collection and $\mathbf{F} = (u, \alpha), (v_1, e_1, C_1), \ldots, (v_k, e_k, C_k)$ be a Vizing fan. We say that the Vizing fan $\mathbf{F}$ is* $\mathcal{U}$-avoiding *if $C_i \subseteq \mathsf{miss}_\chi(v_i) \setminus C_{\mathcal{U}}(v_i)$ for each $v_i \in \mathbf{F}$.*

Using these new definitions, we can directly extend Lemmas 7.1.8 and 7.1.9 to multigraphs by slightly changing the proofs, giving us the following lemmas.

Lemma 7.7.5. *Given a u-edge $\boldsymbol{e} \in \mathcal{U}$, there exists a $\mathcal{U}$-avoiding Vizing fan $\mathbf{F}$ of $\boldsymbol{e}$. Furthermore, we can compute such a Vizing fan in $O(\Delta)$ time.*[15]

Lemma 7.7.6. *Let χ be a $(\Delta + \mu)$-edge coloring of a graph G and $\mathcal{U}$ be a separable collection. For any u-edge $\boldsymbol{e} = (u, v, \alpha) \in \mathcal{U}$ with a $\mathcal{U}$-avoiding Vizing fan $\mathbf{F}$, we have the following:*

1. *Rotating colors around $\mathbf{F}$ does not damage any u-component in $\mathcal{U} \setminus \{\boldsymbol{e}\}$.*
2. *Calling $\mathsf{Vizing}(\mathbf{F})$ damages at most one u-component in $\mathcal{U} \setminus \{\boldsymbol{e}\}$. Furthermore, we can identify the damaged u-component in $O(1)$ time.*

7.7.2 Proof of Theorem 7.7.1

Our main algorithm for multigraphs is completely analogous to our main algorithm for simple graphs, which we describe in Section 7.2. In particular, the algorithm also consists of two main components that are combined in the same way, which we summarize in the following lemmas.

Lemma 7.7.7 (A modification of Lemma 7.2.2 for multigraphs)**.** *There is an algorithm that, given a multigraph G, a partial $(\Delta + \mu)$-edge coloring χ of G and a set of λ uncolored edges U, does one of the following in $O(m + \Delta\lambda)$ time:*

1. *Extends the coloring to $\Omega(\lambda)$ uncolored edges.*
2. *Modifies χ to obtain a separable collection of $\Omega(\lambda)$ u-fans $\mathcal{U}$.*

The proof of this lemma is similar to the proof of Lemma 7.2.2 but with some minor changes. In Section 7.7.3, we sketch how to modify the proof of Lemma 7.2.2 to extend it to Lemma 7.7.7.

[15] Recall that we say that $\mathbf{F}$ is a Vizing fan of $\boldsymbol{e} = (u, v, \alpha)$ if $\mathbf{F}$ is α-primed, has center u, and its first leaf is v.

Lemma 7.7.8 (A modification of Lemma 7.2.3 for multigraphs)**.** *There is an algorithm that, given a multigraph G, a partial $(\Delta+\mu)$-edge coloring χ of G and a separable collection of λ u-fans $\mathcal{U}$, extends χ to $\Omega(\lambda)$ edges in $O(m \log n)$ time w.h.p.*

The proof of this lemma is almost identical to the proof of Lemma 7.2.3, thus we omit the details.

Using Lemmas 7.7.7 and 7.7.8, we get the following lemma which allows us to extend an edge coloring χ to a set of uncolored edges.

Lemma 7.7.9 (A modification of Lemma 7.2.4 for multigraphs)**.** *There is an algorithm that, given a multigraph G and a partial $(\Delta+\mu)$-edge coloring χ of G with λ uncolored edges U, extends χ to the remaining uncolored edges in time $O((m+\Delta\lambda)\log^2 n)$ w.h.p.*

The proof of this lemma is verbatim the same as the proof of Lemma 7.2.4, and hence is omitted.

Finally, in the same way as in the proof of Theorem 7.0.1, we can apply standard Euler partitioning using Lemma 7.7.9 to merge the colorings. We only note that in multigraphs, splitting an m-edge multigraph G via the Euler partition leads to two multigraphs, each with $\lceil m/2 \rceil$ edges, maximum degree $\lceil \Delta/2 \rceil$, and maximum multiplicity $\lceil \mu/2 \rceil$. Thus, the number of colors used to color the graph recursively will be $2 \cdot (\lceil \Delta/2 \rceil + \lceil \mu/2 \rceil) = \Delta + \mu + O(1)$ colors. Hence, we again only need to uncolor $O(m/\Delta)$ edges to obtain a $(\Delta+\mu)$-coloring and then extend this coloring to the remaining $O(m/\Delta)$ edges using Lemma 7.7.9. This leads to an $O(m \log^3 n)$ time algorithm for finding a $(\Delta+\mu)$-edge coloring with high probability.

We can also optimize this algorithm to run in $O(m \log n)$ time with high probability exactly as in our $(\Delta+1)$-edge coloring algorithm. Specifically, using exactly the same argument as in Section 7.6, we have the following analogues of Lemma 7.6.4 and Lemma 7.6.5 for multigraphs.

Lemma 7.7.10 (A modification of Lemma 7.6.4 for multigraphs)**.** *There is an algorithm that, given a multigraph G, finds a $(\Delta+\mu)$-edge coloring of all but (exactly) m/Δ edges in $O(m \log \Delta)$ time with high probability.*

Lemma 7.7.11 (A modification of Lemma 7.6.5 for multigraphs)**.** *There is an algorithm that, given a multigraph G, a partial $(\Delta+\mu)$-edge coloring χ with $\lambda = m/\Delta$ uncolored edges, extends the coloring to all edges in $O(m \log n)$ time with high probability.*

Theorem 7.7.1 now follows immediately from Lemma 7.7.10 and Lemma 7.7.11.

7.7.3 Proof Sketch of Lemma 7.7.7

The algorithm for Lemma 7.7.7 is almost identical to the algorithm for Lemma 7.2.2, except that we replace the relevant definitions with their generalizations for multigraphs, leading to a few extra cases. The overall structure of the algorithm is still the same as Algorithm 7, but using modifications of Prune-Vizing-Fans and Reduce-U-Edges for multigraphs.

Initializing the Seperable Collection $\mathcal{U}$. Given a set of λ uncolored edges U, we can construct a separable collection of λ u-edges $\mathcal{U}$ in the exact same way as simple graphs. However, there may exist distinct u-edges $\boldsymbol{e}, \boldsymbol{e}' \in \mathcal{U}$ that correspond to parallel (but distinct) uncolored edges.

Modifying Prune-Vizing-Fans for Multigraphs. Suppose that we want to run Prune-Vizing-Fans on a separable collection $\mathcal{U}$ with color α. In this call to the subroutine, we consider all of the α-primed u-edges in $\mathcal{E}_\alpha(\mathcal{U}) \subseteq \mathcal{U}$. We implement this subroutine for multigraphs in the same way as for simple graphs, *but with the following additional preprocessing step to ensure that none of the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ are parallel.*

Suppose that we have distinct $\boldsymbol{e}, \boldsymbol{e}' \in \mathcal{E}_\alpha(\mathcal{U})$ that correspond to parallel uncolored edges with endpoints u and v. Then, by the definition of a separable collection, one of these u-edges must have u as a center and the other must have v as a center. Thus, $\alpha \in \mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$, so we can extend the coloring to one of these u-edges by coloring it with α and then remove them both from $\mathcal{U}$. We can scan through all of the α-primed u-edges and perform this operation whenever we encounter parallel u-edges. This maintains the invariant that $\mathcal{U}$ is a separable collection and ensures that none of the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ are parallel.

After performing this preprocessing step, we can implement the rest of Prune-Vizing-Fans in the exact same way as described in Section 7.3.1, except that we must use the generalization of Vizing fans for multigraphs. In particular, all of the lemmas in Section 7.3.1 describing the properties of Prune-Vizing-Fans still hold.

Modifying Reduce-U-Edges for Multigraphs. Suppose that we want to run Reduce-U-Edges on a separable collection $\mathcal{U}$ with color α and have obtained a collection of vertex-disjoint $\mathcal{U}$-avoiding Vizing fans $\mathcal{F}$ corresponding to the u-edges in $\mathcal{E}_\alpha(\mathcal{U})$ from calling Prune-Vizing-Fans. We implement the subroutine Reduce-U-Edges in the same way as described in Section 7.3.2 but with the following modification to Algorithm 8 to account for an additional case.

Suppose that we call Update-Path($\mathbf{F}$) and take one more step along the path `Vizing-Path`($\mathbf{F}$) and observe the edge $e = (x, y)$. If the edge e has already been seen (i.e. is contained in S) then we proceed as normal.[16] If neither the edge e nor any edge parallel to e is contained in S, then we again proceed as normal. However, if we observe that e is not already contained in S but some edge e' that is parallel to e is contained in S, then we do the following: Let $\mathbf{F}' \in \mathcal{F}$ be the Vizing fan such that $e' \in P_{\mathbf{F}'}$, set $\chi(e) \leftarrow \perp$ and $\chi(e') \leftarrow \perp$, call Vizing($\mathbf{F}$) and Vizing($\mathbf{F}'$), set $\chi(e) \leftarrow \alpha$, remove $\mathbf{F}$ and $\mathbf{F}'$ from $\mathcal{F}$ and $\boldsymbol{e}_{\mathbf{F}}$ and $\boldsymbol{e}_{\mathbf{F}'}$ from $\mathcal{U}$, and set $S \leftarrow S \setminus (P_{\mathbf{F}} \cup P_{\mathbf{F}'})$; see Figure 7.7 for an illustration. We can verify that, if this does happen, the edges e and e' must appear in different orientations in the paths $P_{\mathbf{F}}$ and $P_{\mathbf{F}'}$. Consequently, this operation removes 2 u-edges from $\mathcal{U}$ and extends the coloring to one more edge.

After modifying Algorithm 8 with this additional case, we strengthen Invariant 7.3.5 so that no two edges contained in distinct prefix paths in $\{P_{\mathbf{F}}\}_{\mathbf{F} \in \mathcal{F}}$ are parallel. This is sufficient to ensure that the analysis of Reduce-U-Edges in Section 7.3.2 extends to multigraphs. In particular, all of the lemmas in Section 7.3.2 describing the properties of Reduce-U-Edges still hold.

[16] That is, this exact copy of the edge is in S, not just an edge parallel to e.

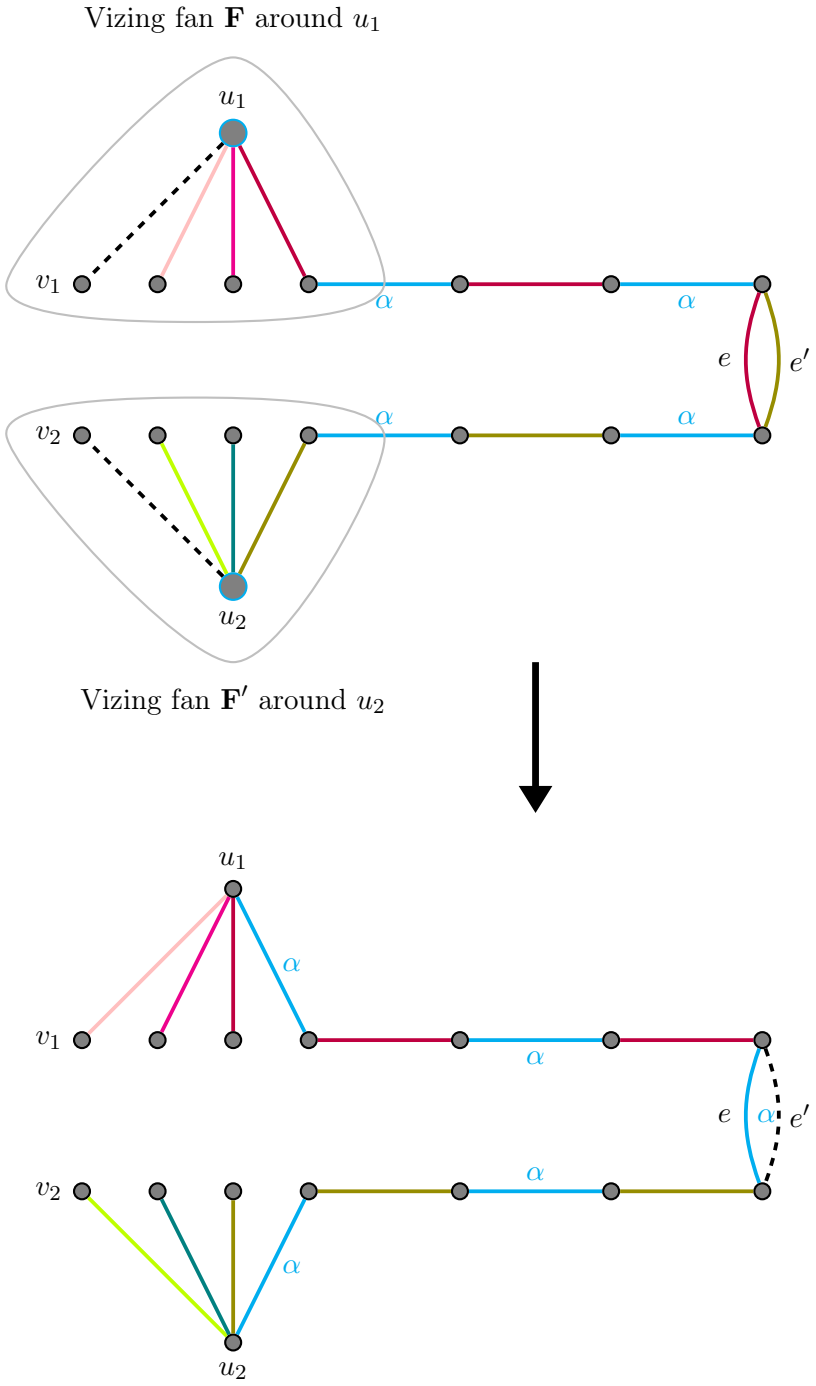

Figure 7.7: If $P_{\mathbf{F}}$ and $P_{\mathbf{F}'}$ meet in the middle at a pair of parallel edges e, e', then we can shift the uncolored edge from $\mathbf{F}'$ to e' and apply $\mathsf{Vizing}(F)$ which assigns $\chi(e) \leftarrow \alpha$; here α is blue.

7.8 Edge Coloring Algorithms for Small Values of Δ

The main contribution and the primary message of our work is presenting the first near-linear time algorithm for $(\Delta+1)$-coloring and polynomial time factor improvements over prior work. However, it turns out that a simple corollary of our main results, plus the prior work of [BD23], is also enough to obtain quite efficient algorithms in the regime when $\Delta = n^{o(1)}$, namely, an algorithm with only $O(m \log \Delta)$ runtime with high probability (basically, replacing $\log n$-term in our main results with a $\log \Delta$-term instead). In this regime, previously, the best known bounds where $O(m\Delta^4 \log \Delta)$ time randomized algorithm of [BD24] for $(\Delta+1)$-edge coloring and $O(m \log \Delta)$ time randomized algorithm of [Ass25] for $(\Delta + O(\log n))$-edge coloring.

Corollary 7.8.1. *There are randomized algorithms that, given any undirected multigraph G on n vertices and m edges with maximum degree Δ and maximum edge multiplicity μ, in $O(m \log \Delta)$ time, with high probability, output:*

1. *a $(\Delta+1)$-coloring of any simple graph G (Vizing's theorem for simple graphs);*
2. *a $(\Delta+\mu)$-coloring of any multigraph G (Vizing's theorem for multigraphs);*

Corollary 7.8.1 improves the runtime dependence of Theorems 7.0.1 and 7.7.1 from $O(m \log n)$ to $O(m \log \Delta)$. In the following, we prove the first part of this result, namely, for $(\Delta+1)$-coloring of simple graphs, and then show how this argument extends to the other two parts as well.

7.8.1 Proof of Corollary 7.8.1 for Simple Graphs

Recall that by Lemma 7.6.4, we already have an algorithm that, with high probability, in $O(m \log \Delta)$ time, $(\Delta+1)$ colors all but m/Δ edges of the graph. Thus, our task is now to extend the coloring to these edges as well. Previously, in Lemma 7.6.5, we did this by coloring a constant fraction of the edges repeatedly (using Lemma 7.6.2); since, we need $\Theta(\log(m/\Delta))$ attempts to color all edges and each one requires $O(m)$ time, this lead to an $O(m \log n)$ time algorithm.

To prove Corollary 7.8.1, we split this final task: we will again run the algorithm of Lemma 7.6.2, but this time to reduce the number of uncolored edges from m/Δ to $m/\operatorname{poly}(\Delta)$ for some arbitrarily large polynomial in Δ; thus, we need $\Theta(\log((m/\Delta)/(m/\operatorname{poly}(\Delta)))) = \Theta(\log \Delta)$ steps of coloring, and hence $O(m \log \Delta)$ time in total. Then, to handle the very last $m/\operatorname{poly}(\Delta)$ remaining edges, we will run the algorithm of [BD24], which extends the coloring to each uncolored edge in essentially (but not formally) $\operatorname{poly}(\Delta)$ time per uncolored edge for some polynomial in Δ. Thus, the second part of the algorithm only takes $O(m)$ time now, leaving us with an $O(m \log \Delta)$ time algorithm all in all. We formalize this approach in the following (which specifically requires extra care for obtaining a high probability guarantee).

Lemma 7.8.2. *Let $c \geq 1$ be any arbitrary fixed constant and $\lambda_0 := \max\{m/\Delta^{2c}, m \log n/n\}$. There is an algorithm that, given a graph G, a partial $(\Delta+1)$-edge coloring χ with $\lambda = m/\Delta$ uncolored edges, extends the coloring to all but $\Theta(\lambda_0)$ uncolored edges in $O(m \log \Delta)$ time with high probability.*

Proof. We simply run our Lemma 7.6.2 with the given parameters λ and λ_0. By Lemma 7.6.2, the expected runtime will be some $T_0 = O(m \log(\lambda/\lambda_0) + \Delta\lambda) = O(m \log \Delta)$ since $\lambda_0 \geq m/\Delta^{2c}$ for some absolute constant $c \geq 1$. Moreover, by the same lemma,

$$\Pr[\text{runtime of the algorithm} \geq 2000 \cdot T_0] \leq \exp(-10\lambda_0) \leq \exp(-\Theta(m \log n)/n) = 1/\operatorname{poly}(n),$$

since $\lambda_0 \geq m \log n/n$. Thus, the algorithm finishes in $O(m \log \Delta)$ time with high probability. □

We now present the final part for coloring the last remaining edges. For this step, we use the following result from [BD23] (itself based on [Ber22, Chr23]) that allows for coloring a random uncolored edge in essentially (but not exactly) $\operatorname{poly}(\Delta)$ time. We note that a recent work of [BD24] presents a further improvement to these bounds (in terms of the exponent of Δ) but as our dependence on Δ is only logarithmic, this improvement is inconsequential and thus we stick with the simpler work of [BD23].

Proposition 7.8.3 ([BD23])**.** *There is an algorithm that, given a graph G, a partial $(\Delta+1)$-edge coloring χ, and λ uncolored edges, extends the coloring to a randomly chosen uncolored edge in expected $\operatorname{poly}(\Delta) \cdot \log(n/\lambda)$ time.*

We use this result to obtain the following lemma for our purpose.

Lemma 7.8.4. *Let $\lambda_0 := \max\{m/\Delta^{2c}, m \log n/n\}$ and let c be the $\operatorname{poly}(\Delta)$-exponent in Proposition 7.8.3. There is an algorithm that, given a graph G, a partial $(\Delta+1)$-edge coloring χ and $\lambda = \Theta(\lambda_0)$ uncolored edges, extends the coloring to all uncolored edges in $O(m \log \Delta)$ time with high probability.*

Proof. Suppose first that the max-term of λ_0 is the second one, i.e., $\lambda_0 = m \log n/n$. This implies that $\Delta \geq (n/\log n)^{1/2c}$ and since c is a constant, we have $\log(\Delta) = \Theta(\log n)$ in this case. We can use the standard Vizing's fan and Vizing's chain approach in Lemma 3.2.2 to color the remaining edges deterministically in $O(\lambda_0 \cdot n) = O(m \log n/n \cdot n) = O(m \log \Delta)$ time in this case.

Now consider the case when $\lambda_0 = m/\Delta^{2c}$. We run the algorithm of Proposition 7.8.3 with the following modification: when coloring a random edge, if the runtime becomes two times more than the expected time guarantee of Proposition 7.8.3, we terminate the coloring and repeat from the beginning. Thus, each step takes $O(\Delta^c \cdot \log(n/i))$ time deterministically and succeeds in extending the coloring by one edge with probability at least half.

For $i \in [\lambda]$, define $\alpha_i := 4 \log(n/i)$ and X_i as the random variable denoting $\ell_i \cdot \alpha_i$ where ℓ_i is the number of times the algorithm attempts to color the next random edge until it succeeds when there are only i edges left uncolored. We first prove that

$$\sum_{i=1}^{\lambda} \alpha_i = o(m/\Delta^c), \tag{7.2}$$

and then show that

$$\Pr\left[\sum_{i=1}^{\lambda} X_i \geq \frac{2m}{\Delta^c}\right] \ll 1/\operatorname{poly}(m), \tag{7.3}$$

which concludes the proof in this case as well since the runtime of coloring the remaining edges is $O(\Delta^{c_0} \cdot \sum_{i=1}^{\lambda} X_i)$ as stated earlier.

Proof of Equation (7.2). We have,

$$\sum_{i=1}^{\lambda} \alpha_i = \sum_{i=1}^{\lambda} 4\log\left(\frac{n}{i}\right) = 4\log\left(\prod_{i=1}^{\lambda} \frac{n}{i}\right) = 4\log\left(\frac{n^\lambda}{\lambda!}\right) \leq \log\left(\left(\frac{e\cdot n}{\lambda}\right)^\lambda\right)$$

(using the inequality $x! \geq (x/e)^x$ for all $x \geq 1$)

$$= 4 \cdot \frac{m}{\Delta^{2c}} \cdot \log\left(\frac{e \cdot n \cdot \Delta^{2c}}{m}\right) = o(m/\Delta^c). \qquad \text{(as } \lambda = m/\Delta^{2c} \text{ and } m \geq n\text{)}$$

Proof of Equation (7.3). Note that for any $i \in [\lambda]$ and $\delta > 0$,

$$\Pr\left[X_i \geq (1+\delta)\cdot\alpha_i\right] = \Pr\left[\ell_i \geq (1+\delta)4\right] \leq 2^{-(3+4\delta)} \leq \exp\left(-\delta\right),$$

where the first inequality is because the probability that ℓ_i increases by one each time is at most $1/2$ by Markov's inequality.

While X_i's are technically not independent (the choice of some edge being colored may change the structure of the graph for coloring next uncolored edge), they are stochastically dominated by independent random variables with above tail bounds (given the guarantee of Proposition 7.8.3 for *each* uncolored edge). Thus, we can apply Proposition 7.6.3 with parameters $\{\alpha_i\}_{i=1}^{\lambda}$ as above and $\beta_i = 1$ for all $i \in [\lambda]$ to obtain that for $t := m/\Delta^c$,

$$\Pr\left[\sum_{i=1}^{\lambda} X_i \geq \frac{2m}{\Delta^c}\right] \leq \Pr\left[\sum_{i=1}^{\lambda} X_i \geq \sum_{i=1}^{\lambda} \alpha_i + t\right] \leq \exp\left(-\frac{1}{4\log n}\cdot\frac{m}{2\Delta^c}\right) \ll 1/\operatorname{poly}(m),$$

where the first two inequalities use Equation (7.2) for $\sum_i \alpha_i$ and the third uses $\Delta^c \leq (n/\log n)^{c/2c} < \sqrt{n}$ in this case (when λ_0 is the first-term of its maximum). This concludes the proof. □

The proof of Corollary 7.8.1 for $(\Delta+1)$-edge coloring is now as follows: run Lemma 7.6.4 to color all but m/Δ edges. Then, let c be the constant in the exponent of $\operatorname{poly}(\Delta)$ in Proposition 7.8.3 and run Lemma 7.8.2 to extend the coloring to all but $\lambda_0 := \max\{m/\Delta^{2c}, m\log n/n\}$ edges. Finally, run Lemma 7.8.4 to extend the coloring to all these remaining λ_0 edges. Since each of these algorithms take $O(m\log\Delta)$ time and succeed with high probability, we obtain the final algorithm.

7.8.2 Proof of Corollary 7.8.1 for Multigraphs

To extend the proof to multigraphs, recall that by Lemma 7.7.10 we can $(\Delta+\mu)$-color all but m/Δ edges of a multigraph in $O(m \log \Delta)$ time. Thus, we can follow the same exact strategy as in the $(\Delta+1)$-coloring approach outlined in the previous subsection, and we only need an analogue of the result of [BD23] to conclude the proof. Such a result are also already established by [Dha24] and we do not need to reinvent the wheel here.

Proposition 7.8.5 (A subroutine in [Dha24, Theorem 1.6; part 1])**.** *There is an algorithm that, given a multigraph G, a partial $(\Delta+\mu)$-edge coloring χ, and λ uncolored edges, extends the coloring to a randomly chosen uncolored edge in expected* $\text{poly}(\Delta) \cdot \log(n/\lambda)$ *time.*

The rest of the proof is verbatim as in simple graphs in Corollary 7.8.1 and we omit it here.

7.9 Shannon's Theorem for Multigraphs in Near-Linear Time

In this section, we show that our near-linear time algorithm can also be extended to Shannon's theorem for multigraphs, proving Theorem 1.1.5, which we restate below.

Theorem 1.1.5. *There is an algorithm that, given a multigraph G with maximum degree Δ, computes a $\lfloor 3\Delta/2 \rfloor$-coloring of G in $O(m \log \Delta)$ time with high probability.*

A simple and direct proof of Shannon's theorem can be obtained by using Vizing's theorem for multigraphs as a black-box [Pos21].[17] We prove Theorem 1.1.5 by showing that this proof gives an efficient reduction from Shannon's theorem to Vizing's theorem in the the static setting.

7.9.1 Reducing Shannon's Theorem to Vizing's Theorem

In this section, we show that any static algorithm that computes a $(\Delta+\mu)$-coloring given a multigraph G with maximum degree Δ and maximum multiplicity μ can be used to compute a $\lfloor 3\Delta/2 \rfloor$-coloring, while only incurring an additive $O(m)$ factor in the running time.

The Reduction. Let $E_{\texttt{heavy}} \subseteq E(G)$ be the edges in G with multiplicity at least $\lceil \Delta/2 \rceil$ (i.e. all edges between pairs of vertices u and v such that there are at least $\lceil \Delta/2 \rceil$ distinct edges with endpoints (u,v)), and let G' be the graph obtained from G by removing all of the edges in $E_{\texttt{heavy}}$. Now, using the $(\Delta+\mu)$-coloring algorithm, compute a $(\Delta(G')+\mu(G'))$-coloring χ' of G'.

Claim 7.9.1. *We have that χ' is a partial $\lfloor 3\Delta/2 \rfloor$-coloring of G.*

Proof. Since $\Delta(G') \leq \Delta$ and $\mu(G') < \lceil \Delta/2 \rceil$, we have that $\Delta(G')+\mu(G') < \Delta + \lceil \Delta/2 \rceil$. Thus, since $\Delta(G')+\mu(G')$ is a integer, we have that $\Delta(G')+\mu(G') \leq \lfloor 3\Delta/2 \rfloor$. □

Now, we proceed to scan through the edges in $E_{\texttt{heavy}}$ and extend the coloring to each of them one by one. Crucially, *we do not need to recolor any of the other edges in G while extending the coloring to some edge $e \in E_{\texttt{heavy}}$.*

[17] We thank David Wajc for bringing this simple proof to our attention [Waj24].

Lemma 7.9.2. *Let χ be a partial $\lfloor 3\Delta/2 \rfloor$-coloring of G and suppose that $e \in E_{\texttt{heavy}}$ is uncolored. Then there exists a color α that is available at both endpoints of e.*

Proof. Let u and v be the endpoints of e and let $\mu(e)$ be the multiplicity of the edge e. Then the total number of colors assigned to edges incident on either u or v is at most

$$(\deg_G(u) - \mu(e)) + (\deg_G(v) - \mu(e)) + (\mu(e) - 1) = \deg_G(u) + \deg_G(v) - \mu(e) - 1$$

$$\leq 2\Delta - \left\lceil \frac{\Delta}{2} \right\rceil - 1 = \left\lfloor \frac{3\Delta}{2} \right\rfloor - 1.$$

Thus, we have that at least $\lfloor 3\Delta/2 \rfloor - (\lfloor 3\Delta/2 \rfloor - 1) \geq 1$ colors are available at both of the vertices u and v. It follows that there must exist some color which is available at e. □

For each edge $e \in E_{\texttt{heavy}}$ with endpoints u and v, there are many other edges in $E_{\texttt{heavy}}$ with the exact same endpoints. Let $E^{u,v}_{\texttt{heavy}}$ denote the edges in $E_{\texttt{heavy}}$ with endpoints u and v. By definition, we have that $|E^{u,v}_{\texttt{heavy}}| \geq \lceil \Delta/2 \rceil = \Omega(\Delta)$. Furthermore, we can explicitly compute the set of colors $\mathsf{miss}_\chi(u) \cap \mathsf{miss}_\chi(v)$ and assign each edge in $E^{u,v}_{\texttt{heavy}}$ one of these colors in $O(\Delta)$ time. Since there are $O(|E_{\texttt{heavy}}|/\Delta)$ such groups, it takes $O(|E_{\texttt{heavy}}|) \leq O(m)$ time to extend the coloring χ' to all of the edges in G, obtaining a $\lfloor 3\Delta/2 \rfloor$-coloring of G.

7.10 A Specialized Concentration Inequality

Throughout the chapter, for optimizing log factors in the runtime of our algorithms, we needed the following specialized concentration inequality. For completeness, we now prove this inequality using a standard argument based on moment generating functions.

Proposition 7.6.3. *Let $\{X_i\}_{i=1}^n$ be n independent non-negative random variables associated with parameters $\{\alpha_i\}_{i=1}^n$ and $\{\beta_i\}_{i=1}^n$ such that for each $i \in [n]$, $\alpha_i, \beta_i \geq 1$, and for every $\delta > 0$,*

$$\Pr\left[X_i \geq (1+\delta) \cdot \alpha_i\right] \leq \exp\left(-\delta \cdot \beta_i\right);$$

then, for every $t \geq 0$,

$$\Pr\left[\sum_{i=1}^n X_i \geq \sum_{i=1}^n \alpha_i + t\right] \leq \exp\left(-\left(\min_{i=1}^n \frac{\beta_i}{2\alpha_i}\right) \cdot \left(t - 2\sum_{i=1}^n \frac{\alpha_i}{\beta_i}\right)\right).$$

Proof. Let $s := 1/2 \cdot \min_i \beta_i/\alpha_i$ and for $i \in [n]$, $Y_i := \max(X_i - \alpha_i, 0)$. Letting $Y := \sum_{i=1}^n Y_i$, leads

$$\Pr\left[\sum_{i=1}^n X_i \geq \sum_{i=1}^n \alpha_i + t\right] \leq \Pr\left[Y \geq t\right],$$

for all $t \geq 0$. We further have,

$$\Pr[Y \geq t] = \Pr\left[e^{s \cdot Y} \geq e^{s \cdot t}\right] \leq e^{-s \cdot t} \cdot \mathbb{E}\left[e^{s \cdot Y}\right] = e^{-s \cdot t} \cdot \mathbb{E}\left[\prod_{i=1}^{n} e^{s \cdot Y_i}\right] = e^{-s \cdot t} \cdot \prod_{i=1}^{n} \mathbb{E}\left[e^{s \cdot Y_i}\right] \tag{7.4}$$

where the inequality is by Markov inequality and the final equality is by the independence of the variables. Now, for every $i \in [n]$,

$$\begin{aligned}
\mathbb{E}\left[e^{s \cdot Y_i}\right] &= \int_{y=0}^{\infty} \Pr[Y_i = y \cdot \alpha_i] \cdot \exp(s \cdot y \cdot \alpha_i)\ dy \\
&= \int_{y=0}^{\infty} \Pr[X_i = (1+y) \cdot \alpha_i] \cdot \exp(s \cdot y \cdot \alpha_i)\ dy \\
&\qquad\qquad \text{(by the connection between } X_i \text{ and } Y_i \text{ when } y \geq 0) \\
&\leq \int_{y=0}^{\infty} \exp(-y \cdot \beta_i) \cdot \exp(s \cdot y \cdot \alpha_i)\ dy \\
&\qquad\qquad \text{(by the tail inequality of } X_i \text{ in the proposition statement)} \\
&= \frac{1}{\beta_i \cdot (1 - s \cdot (\alpha_i/\beta_i))} \\
&\text{(since the integral of } e^{-Kx} = -e^{-Kx}/K \text{ and } s < \beta_i/\alpha_i \text{ thus the value of this function in } \infty \text{ is zero)} \\
&\leq \frac{1}{1 - s \cdot (\alpha_i/\beta_i)} \qquad\qquad \text{(as } \beta_i \geq 1) \\
&\leq \exp(2s \cdot \alpha_i/\beta_i). \qquad\qquad \text{(as } s \cdot \alpha_i/\beta_i < 1/2 \text{ and } 1 - x \geq e^{-2x} \text{ for } x < 1/2)
\end{aligned}$$

Plugging this bounds in Equation (7.4), implies that

$$\Pr[Y \geq t] \leq \exp\left(-s \cdot t + \sum_{i=1}^{n} 2s \cdot \frac{\alpha_i}{\beta_i}\right) = \exp\left(-\left(\min_{i=1}^{n} \frac{\beta_i}{2\alpha_i}\right) \cdot \left(t - 2\sum_{i=1}^{n} \frac{\alpha_i}{\beta_i}\right)\right),$$

by the choice of s, concluding the proof. □

Chapter 8

Vizing's Theorem in Deterministic Almost-Linear Time

In this chapter, we give our full deterministic almost-linear time algorithm for $(\Delta+1)$-coloring, proving Theorem 1.1.2, which we restate below.

Theorem 1.1.2. *There is a deterministic algorithm that, given a graph G with maximum degree Δ, computes a $(\Delta+1)$-coloring of G in $m \cdot 2^{O(\sqrt{\log \Delta})} \cdot \log n = m^{1+o(1)}$ time.*

In Section 8.1 we give the necessary notation and preliminaries. In Section 8.2 we demonstrate how our main result relies on three subroutines; we state three lemmas that describe the behaviour of these subroutines and use them to prove Theorem 1.1.2. The rest of the chapter is devoted to presenting these subroutines and formally analyzing them by proving these lemmas.

Notation. This chapter uses the notation described in Chapter 3. We introduce the rest of the notation used throughout this chapter in Section 8.1. This chapter uses the notion of u-fans and separable collections introduced in Chapter 7, but with minor notational differences for convenience.

8.1 Preliminaries

This section introduces the notation used throughout this chapter. We note that there are minor notational differences with the definitions in Section 7.1.1 for convenience.

8.1.1 U-Fans and Separable Collections

Definition 8.1.1 (u-fan)**.** *A* u-fan *is a tuple $\boldsymbol{f} = (u, v, w, c_{\boldsymbol{f}}(u), c_{\boldsymbol{f}}(v), c_{\boldsymbol{f}}(w))$ where u, v and w are distinct vertices and $c_{\boldsymbol{f}}(u)$, $c_{\boldsymbol{f}}(v)$ and $c_{\boldsymbol{f}}(w)$ are colors such that:*

1. *(u, v) and (u, w) are uncolored edges.*
2. *$c_{\boldsymbol{f}}(u) \in \mathsf{miss}_\chi(u)$, $c_{\boldsymbol{f}}(v) \in \mathsf{miss}_\chi(v)$ and $c_{\boldsymbol{f}}(w) \in \mathsf{miss}_\chi(w)$.*

3. $c_{\boldsymbol{f}}(u) \neq c_{\boldsymbol{f}}(v)$ *and* $c_{\boldsymbol{f}}(v) = c_{\boldsymbol{f}}(w)$.

We say that u is the *center* of $\boldsymbol{f}$ and that v and w are the *leaves* of $\boldsymbol{f}$. Given a vertex $x \in \boldsymbol{f}$, we say that $c_{\boldsymbol{f}}(x)$ is the available color that $\boldsymbol{f}$ 'assigns' to x. We define a **damaged u-fan** in the same way as a u-fan, except that we do not require Condition 3 in Definition 8.1.1.

Color Types. Given a set of colors C, we define a **type** as an unordered pair of colors in C. Given $A, B \subseteq C$, we abuse notation slightly and let $A \times B$ denote the set of types with one color in A and one color in B, i.e. the set $\{\{\alpha, \beta\} \mid \alpha \in A, \beta \in B\}$.

Given a u-fan $\boldsymbol{f}$, we define *the type of* $\boldsymbol{f}$ as $\tau(\boldsymbol{f}) := \{c_{\boldsymbol{f}}(x)\}_{x \in \boldsymbol{f}}$. Note that $|\tau(\boldsymbol{f})| = 2$, so $\tau(\boldsymbol{f})$ satisfies the definition of a type. For a damaged u-fan $\boldsymbol{f}$, we define $\tau(\boldsymbol{f})$ in the same way, but it is not necessarily a type since we might have $|\tau(\boldsymbol{f})| = 1$ or $|\tau(\boldsymbol{f})| = 3$. For notational convenience, we still refer to $\tau(\boldsymbol{f})$ as the type of $\boldsymbol{f}$ even if $\boldsymbol{f}$ is a damaged u-fan.

Activating U-Fans. Let $\boldsymbol{f}$ be a u-fan with center u, leaves v and w, and type $\tau(\boldsymbol{f}) = \{\alpha, \beta\}$ such that $c_{\boldsymbol{f}}(u) = \alpha$ and $c_{\boldsymbol{f}}(v) = c_{\boldsymbol{f}}(w) = \beta$. The *key property* of u-fans is that at most one of the $\{\alpha, \beta\}$-alternating paths starting at v or w ends at u. Suppose that the $\{\alpha, \beta\}$-alternating path starting at v does not end at u. Then, after flipping this $\{\alpha, \beta\}$-alternating path, both vertices u and v are missing color α. Thus, we can extend the coloring χ by assigning $\chi(u, v) \leftarrow \alpha$. We refer to this as *activating* the u-fan $\boldsymbol{f}$.

Collections of U-Fans. Throughout this chapter, we consider collections of u-fans $\mathcal{U}$. We only use the term 'collection' in this context, so we abbreviate 'collection of u-fans' by just 'collection'.

Definition 8.1.2 (Separable Collection)**.** *Let χ be a partial μ-coloring of G with colors C and $\mathcal{U}$ be a collection of u-fans. We say that the collection $\mathcal{U}$ is* separable *if the following holds:*

1. *All u-fans in $\mathcal{U}$ are edge-disjoint.*
2. *For each $x \in V$, the colors in the multi-set $C_{\mathcal{U}}(x) := \{c_{\boldsymbol{f}}(x) \mid \boldsymbol{f} \in \mathcal{U}, x \in \boldsymbol{f}\}$ are distinct.*

8.1.2 Constructing Separable Collections

To construct separable collections of u-fans, we use a fast deterministic algorithm given described in Lemma 7.2.2 which takes a set U of λ uncolored edges and either extends the coloring to a constant fraction of the edges in U or shifts these uncolored edges around the graph in order to construct a separable collection of $\Omega(\lambda)$ u-fans. We restate Lemma 7.2.2 below.

Lemma 8.1.3. *Given a graph G, a partial $(\Delta+1)$-coloring χ of G and a set of λ uncolored edges U, there is a deterministic algorithm that does one of the following in $O((m + \Delta\lambda)\log\Delta)$ time:*

1. *Extends the coloring to $\Omega(\lambda)$ uncolored edges.*
2. *Modifies χ to obtain a separable collection of $\Omega(\lambda)$ u-fans $\mathcal{U}$.*

We remark that the algorithm described by Lemma 8.1.3 crucially uses Vizing fans and chains. The rest of our algorithm does not use Vizing fans and chains.

8.1.3 Alternating Paths in Separable Collections

The main way that our algorithms modify the coloring χ of a graph is by flipping alternating paths. However, our algorithms will also often explicitly maintain a separable collection of u-fans $\mathcal{U}$ while modifying the coloring χ, and we want to ensure that $\mathcal{U}$ remains separable at all times. Thus, whenever we flip an alternating path, we need to appropriately modify the missing colors assigned to (and hence also the types of) some u-fans. We do this by using the following simple subroutine Flip-Path to flip alternating paths.

The Algorithm Flip-Path**.** As input, the subroutine Flip-Path is given a graph G, a coloring χ of G with colors C, a separable collection $\mathcal{U}$, and a maximal $\{\alpha, \beta\}$-alternating path P. The subroutine then flips the colors of the alternating path P and modifies the missing colors assigned to the vertices of some u-fans in $\mathcal{U}$ to ensure that $\mathcal{U}$ remains separable. The pseudocode in Algorithm 12 gives a formal description of how we implement the procedure.

Algorithm 12: Flip-Path(P)

```
Let x and y be the endpoints of P
Flip the colors of the {α, β}-alternating path P
for z ∈ {x, y} do
    if there exists f ∈ U s.t. c_f(z) ∈ {α, β} then
        Let c ∈ {α, β} \ {c_f(z)}
        Set c_f(z) ← c
```

Lemma 8.1.4. *For any maximal alternating path P, the collection $\mathcal{U}$ remains separable after applying* Flip-Path(P)*. Furthermore, this damages at most 2 u-fans.*

8.1.4 Data Structures

In Section 8.6, we describe the data structures that we use to implement our algorithm. Our data structures are simple and almost identical to the data structures in Section 7.5, with the modification that we use balanced binary trees instead of hashmaps to make our data structures deterministic, incurring an extra $O(\log \mu)$ factor overhead in the running time of each operation, where μ is the number of colors in the edge coloring χ. For the sake of completeness, we present the full deterministic data structures.

On top of the standard data structures used to maintain the μ-coloring χ with colors C, we also use data structures that allow us to efficiently maintain a separable collection $\mathcal{U}$. More specifically, the data structures that we use to implement a separable collection $\mathcal{U}$ support the following queries.

- Insert$_{\mathcal{U}}(\boldsymbol{f})$: The input to this query is a u-fan $\boldsymbol{f}$. In response, the data structure adds $\boldsymbol{f}$ to $\mathcal{U}$ if $\mathcal{U} \cup \{\boldsymbol{f}\}$ is separable and outputs `fail` otherwise.
- Delete$_{\mathcal{U}}(\boldsymbol{f})$: The input to this query is a u-fan $\boldsymbol{f}$. In response, the data structure removes $\boldsymbol{f}$ from $\mathcal{U}$ if $\boldsymbol{f} \in \mathcal{U}$ and outputs `fail` otherwise.

- FIND-U-FAN$_{\mathcal{U}}(x, c)$: The input to this query is a vertex $x \in V$ and a color $c \in C$. In response, the data structure returns the u-fan $\boldsymbol{f} \in \mathcal{U}$ with $c_{\boldsymbol{f}}(x) = c$ if such a u-fan exists and outputs `fail` otherwise.

- MISSING-COLOR$_{\mathcal{U}}(x)$: The input to this query is a vertex $x \in V$. In response, the data structure returns an arbitrary color from the set $\mathsf{miss}_{\chi}(x) \setminus C_{\mathcal{U}}(x)$.

Furthermore, the data structure supports the following initialization operation.

- INITIALIZE(G, χ): Given a graph G and an edge coloring χ of G, we can initialize the data structure with an empty separable collection $\mathcal{U} = \emptyset$.

In Section 8.6, we show how to implement the initialization operation in $O(m \log \mu)$ time and each of these queries in $O(\log \mu)$ time with the appropriate data structures. **These queries provide the 'interface' via which our algorithm will interact with the u-components.**

8.2 The Main Algorithm: Proof of Theorem 1.1.2

Our final algorithm can be easily described using three different deterministic subroutines as black boxes, which will be presented in subsequent sections. In this section, we state lemmas that summarize the behavior of these subroutines and show how to use them to prove Theorem 1.1.2.

The first of these subroutines and the main technical component of our algorithm is a subroutine called Sparsify-Types, which is described in Section 8.3. The following lemma (proved in Section 8.3) summarizes the behavior of Sparsify-Types.

Lemma 8.2.1. *Given a graph G, a partial μ-coloring χ of G with colors C, a separable collection $\mathcal{U}$ of size λ, and an integer $10 \leq \eta \leq \mu/10$, the algorithm* Sparsify-Types *modifies the coloring χ (without changing which edges are colored) and the separable collection $\mathcal{U}$, and constructs disjoint subsets of colors $\mathcal{C}_1, \ldots, \mathcal{C}_\eta \subseteq C$ with the following properties:*

1. *For all $k \in [\eta]$, we have that $|\mathcal{C}_k| \leq \mu/\eta$.*
2. *The u-fans in $\mathcal{U}$ have types in $\bigcup_{k=1}^{\eta}(\mathcal{C}_k \times \mathcal{C}_k)$ and $|\mathcal{U}| \geq \lambda/100$.*

Furthermore, the algorithm is deterministic and runs in time $O(m\eta^4 \log \mu)$.

The second subroutine, described in Section 8.4, is called Color-Small, and we use it to extend the coloring to sufficiently small subgraphs. The following lemma (proved in Section 8.4) summarizes the behavior of Color-Small.

Lemma 8.2.2. *Given a graph G, a partial μ-coloring χ with colors C, and a separable collection $\mathcal{U}$ of size λ, the algorithm* Color-Small *extends the coloring χ to at least $\lambda/100$ uncolored edges. Furthermore, the algorithm is deterministic and runs in $O(m\mu^2 \log \mu)$ time.*

Using the algorithms Sparsify-Types and Color-Small, we construct a *recursive* algorithm Extend-Coloring, described in Section 8.5, which takes a partial coloring as input and efficiently extends the coloring to a large proportion of the uncolored edges. The following lemma (proved in Section 8.5) summarizes the behavior of Extend-Coloring.

Lemma 8.2.3. *Given a graph G, a partial μ-coloring χ of G with colors C, a separable collection $\mathcal{U}$ of size λ, and an integer $\eta \geq 10$, the algorithm* Extend-Coloring *extends the coloring χ to at least $\lambda/100^{(\log \mu/\log \eta)+1}$ uncolored edges. Furthermore, the algorithm is deterministic and runs in time $O(m\eta^4 \log^2(\mu)/\log(\eta))$.*

Using Lemma 8.2.3, we can now prove Theorem 1.1.2.

8.2.1 Proof of Theorem 1.1.2

We begin by proving the following corollary of Lemma 8.2.3, which we use to efficiently extend a $(\Delta+1)$-coloring of a graph G.

Corollary 8.2.4. *Given a graph G and a partial $(\Delta+1)$-coloring χ of G with $\lambda = O(m/\Delta)$ uncolored edges U, we can extend χ to the remaining uncolored edges in time $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n)$.*

Proof. Suppose that we have a partial $(\Delta+1)$-coloring χ with λ uncolored edges. Applying Lemma 8.1.3, we either extend the coloring χ to $\Omega(\lambda)$ uncolored edges or modify χ to obtain a separable collection of $\Omega(\lambda)$ u-fans $\mathcal{U}$ in $O(m)$ time. In the latter case, we can then apply Lemma 8.2.3 to the coloring χ and the separable collection $\mathcal{U}$ with $\eta = 10 \cdot \lceil 2^{\sqrt{\log_2 \Delta}} \rceil$ to extend the coloring χ to an $\Omega(1/100^{\log \Delta/\log \eta+1}) \geq \Omega(1/100^{\sqrt{\log_2 \Delta}})$ proportion of the uncolored edges. We can repeat this $O(100^{\sqrt{\log_2 \Delta}} \log \lambda)$ times to extend the coloring χ to all of the edges in U. Each iteration of this process can be implemented in $O(m \cdot 2^{4\sqrt{\log_2 \Delta}} \log^2 \Delta) \leq O(m \cdot 2^{6\sqrt{\log_2 \Delta}})$ time, since we repeat this process for $O(100^{\sqrt{\log_2 \Delta}} \log \lambda) \leq O(2^{7\sqrt{\log_2 \Delta}} \log n)$ iterations, this takes $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n)$ time in total. □

We prove Theorem 1.1.2 by applying Corollary 8.2.4 to the standard Euler partition framework [GNK+85, Sin19]. Given a graph G, we partition it into two edge-disjoint subgraphs G_1 and G_2 on the same vertex set such that $\Delta(G_i) \leq \lceil \Delta/2 \rceil$ for each G_i, where $\Delta(G_i)$ denotes the maximum degree of G_i. We then recursively compute a $(\Delta(G_i)+1)$-coloring χ_i for each G_i. Combining χ_1 and χ_2, we obtain a $(\Delta+3)$-coloring χ of G. We then uncolor the two smallest color classes in χ, which contain $O(m/\Delta)$ edges, and apply Corollary 8.2.4 to recolor all of the uncolored edges in χ using only $\Delta+1$ colors in $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n)$ time.

To show that the total running time of the algorithm is $O(m \cdot 2^{14\sqrt{\log_2 \Delta}} \log n)$, first note that the depth of the recursion tree is $O(\log \Delta)$. Next, consider the i^{th} level of the recursion tree, for an arbitrary $i = O(\log \Delta)$: we have 2^i edge-disjoint subgraphs $G_1, \ldots, G_{2^i}$ such that $\Delta(G_j) \leq O(\Delta/2^i)$ and $\sum_{j=1}^{2^i} |E(G_j)| = m$. Since the total running time at recursion level i is $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n)$ and the depth of the recursion tree is $O(\log \Delta)$, it follows that the total running time is $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n \log \Delta) \leq O(m \cdot 2^{14\sqrt{\log_2 \Delta}} \log n)$.

8.3 The Algorithm Sparsify-Types: Proof of Lemma 8.2.1

In this section, we describe and analyze the subroutine Sparsify-Types, proving Lemma 8.2.1, which we restate below.

Lemma 8.2.1. *Given a graph G, a partial μ-coloring χ of G with colors C, a separable collection $\mathcal{U}$ of size λ, and an integer $10 \leq \eta \leq \mu/10$, the algorithm* Sparsify-Types *modifies the coloring χ (without changing which edges are colored) and the separable collection $\mathcal{U}$, and constructs disjoint subsets of colors $\mathcal{C}_1, \ldots, \mathcal{C}_\eta \subseteq C$ with the following properties:*

1. *For all $k \in [\eta]$, we have that $|\mathcal{C}_k| \leq \mu/\eta$.*
2. *The u-fans in $\mathcal{U}$ have types in $\bigcup_{k=1}^{\eta}(\mathcal{C}_k \times \mathcal{C}_k)$ and $|\mathcal{U}| \geq \lambda/100$.*

Furthermore, the algorithm is deterministic and runs in time $O(m\eta^4 \log \mu)$.

8.3.1 Preliminaries for Sparsify-Types

We begin by giving some preliminaries and defining notations and subroutines that we use to describe the algorithm Sparsify-Types.

The Subsets of Colors $C_1, \ldots, C_{2\eta}$. Let G be a graph, $\chi : E(G) \longrightarrow C \cup \{\perp\}$ be a partial μ-coloring of G, $\mathcal{U}$ a separable collection with types in $C \times C$, and η an integer such that $10 \leq \eta \leq \mu/10$. By relabeling the colors, we can assume that $C = [\mu]$. Let $r := \lfloor \mu/2\eta \rfloor$, $C_i := [(i-1)r + 1, ir]$ for each $i \in [2\eta]$, and $\mathcal{C}_k := C_{2k-1} \cup C_{2k}$ for each $k \in [\eta]$. Note that $|\mathcal{C}_k| = 2r \leq \mu/\eta$ for each $k \in [\eta]$. For each $i \in [2\eta]$ and $j \in [r]$, we denote the j^{th} color in C_i by $C_i(j) := (i-1)r + j$. We can see that the subsets of colors $C_1, \ldots, C_{2\eta} \subseteq [\mu]$ partition the set of colors $[2\eta r] \subseteq [\mu]$.

Our algorithm will modify χ by only changing the colors of edges $e \in E$ with $\chi(e) \in [q]$, where $q := 2\eta r$. Thus, our algorithm will only have the ability to change the types of u-fans $\boldsymbol{f} \in \mathcal{U}$ with $\tau(\boldsymbol{f}) \in [q] \times [q]$.[1] To ensure that sufficiently many u-fans have types in $[q] \times [q]$, we perform a simple preprocessing step that involves relabeling the colors; we describe this in the proof of Claim 8.3.2. After this preprocessing step, our algorithm constructs the subsets of colors $C_1, \ldots, C_{2\eta}$ and removes all u-fans $\boldsymbol{f}$ from $\mathcal{U}$ that do *not* have a type $\tau(\boldsymbol{f}) \in [q] \times [q]$. Thus, from this point onward, *we assume that all u-fans in $\mathcal{U}$ have types in $\tau(\boldsymbol{f}) \in [q] \times [q]$*. We later show that this operation only removes at most $2\lambda/5$ u-fans from $\mathcal{U}$.

Uniform and Aligned U-Fans. Let $\boldsymbol{f}$ be a u-fan in $\mathcal{U}$. We say that the u-fan $\boldsymbol{f}$ is **uniform** if $\tau(\boldsymbol{f}) \subseteq C_i$ (i.e. $\tau(\boldsymbol{f}) \in C_i \times C_i$) for some $i \in [2\eta]$, and we say that $\boldsymbol{f}$ is **aligned** if $\tau(\boldsymbol{f})$ intersects both C_{2k-1} and C_{2k} (i.e. $\tau(\boldsymbol{f}) \in C_{2k-1} \times C_{2k}$) for some $k \in [\eta]$. We refer to u-fans that are uniform or aligned as **social**. We let $\hat{\mathcal{U}} \subseteq \mathcal{U}$ denote the subset of social u-fans. Note that for any social fan $\boldsymbol{f} \in \hat{\mathcal{U}}$, we have that $\tau(\boldsymbol{f}) \in \bigcup_{i=1}^{\eta}(\mathcal{C}_i \times \mathcal{C}_i)$. Thus, our objective is to ensure that a constant fraction of the u-fans in $\mathcal{U}$ are social, i.e. that $|\hat{\mathcal{U}}| = \Omega(\lambda)$.

[1] Recall that $[q] \times [q]$ denotes the set of *unordered* subsets $\{\{c, c'\} \mid c, c' \in [q]\}$.

Modifying the Types of U-Fans. The algorithm Sparsify-Types works by repeatedly modifying the types of u-fans to increase the number of social u-fans. Suppose that the algorithm wants to modify the type of a u-fan $\boldsymbol{f} \notin \hat{\mathcal{U}}$ with $\tau(\boldsymbol{f}) \in C_i \times C_{i'}$ (by changing the colors of some edges) so that $\tau(\boldsymbol{f}) \in \mathcal{C}_k \times \mathcal{C}_k$ for some $k \in [\eta]$ after the modification. The algorithm does this by flipping some specific alternating paths, that we refer to as the k-**relevant** alternating paths of $\boldsymbol{f}$, which changes the type of $\boldsymbol{f}$ so that it is aligned. Suppose that $\boldsymbol{f} = (u, v, w, c_{\boldsymbol{f}}(u), c_{\boldsymbol{f}}(v), c_{\boldsymbol{f}}(w))$ where $c_{\boldsymbol{f}}(u) = C_i(j)$ and $c_{\boldsymbol{f}}(v) = c_{\boldsymbol{f}}(w) = C_{i'}(j')$ for $i, i' \in [2\eta]$ and $j, j' \in [r]$. Note that, since $\boldsymbol{f}$ is not social (and hence not uniform) we have that $i \neq i'$. In the case that $\{2k-1, 2k\} \cap \{i, i'\} = \emptyset$, the k-relevant alternating paths of $\boldsymbol{f}$ are

1. The $\{C_{2k-1}(j), C_i(j)\}$-alternating path P_u starting at u.
2. The $\{C_{2k}(j'), C_{i'}(j')\}$-alternating paths P_v and P_w starting at v and w respectively.

We let $\mathcal{P}_k(\boldsymbol{f})$ denote the set of alternating-paths $\{P_u, P_v, P_w\}$.[2] If $i = 2k$ or $i' = 2k-1$, we consider the $\{C_{2k}(j), C_i(j)\}$-alternating path starting at u and the $\{C_{2k-1}(j'), C_{i'}(j')\}$-alternating paths starting at v and w instead. Flipping the alternating paths in $\mathcal{P}_k(\boldsymbol{f})$ by calling Flip-Path(P) for each $P \in \mathcal{P}_k(\boldsymbol{f})$ turns the type of the u-fan $\boldsymbol{f}$ into $\{C_{2k-1}(j), C_{2k}(j')\} \in C_{2k-1} \times C_{2k} \subseteq \mathcal{C}_k \times \mathcal{C}_k$. We note that flipping these paths might also change the types of at most 3 other u-fans in the collection $\mathcal{U}$, *potentially damaging these u-fans*. Algorithm 13 provides the pseudocode for the algorithm `Compute-Paths` that computes the set of k-relevant alternating paths of $\boldsymbol{f}$.

Algorithm 13: `Compute-Paths`$(\boldsymbol{f}, k)$

Let $\boldsymbol{f} = (u, v, w, c_{\boldsymbol{f}}(u), c_{\boldsymbol{f}}(v), c_{\boldsymbol{f}}(w))$
Let $c_{\boldsymbol{f}}(u) = C_i(j)$ and $c_{\boldsymbol{f}}(v) = c_{\boldsymbol{f}}(w) = C_{i'}(j')$, where $i, i' \in [2\eta]$ and $j, j' \in [r]$
Let $\ell \leftarrow 2k-1$ and $\ell' \leftarrow 2k$
if $i = 2k$ *or* $i' = 2k-1$ **then**
  Let $\ell \leftarrow 2k$ and $\ell' \leftarrow 2k-1$
Let P_u be the $\{C_\ell(j), C_i(j)\}$-alternating path starting at u
Let P_v and P_w be the $\{C_{\ell'}(j'), C_{i'}(j')\}$-alternating path starting at v and w respectively
return $\{P_u, P_v, P_w\}$

For $i, i' \in [2\eta]$, let $\mathcal{U}_{i,i'} := \{\boldsymbol{f} \in \mathcal{U} \mid \tau(\boldsymbol{f}) \in C_i \times C_{i'}\} \subseteq \mathcal{U}$ denote the subset of u-fans with types in $C_i \times C_{i'}$. A key observation is that the k-relevant alternating paths corresponding to the u-fans in $\mathcal{U}_{i,i'} \setminus \hat{\mathcal{U}}$ are either edge-disjoint or the same. We summarize this property in the following claim.

Claim 8.3.1. *For any u-fans $\boldsymbol{f}, \boldsymbol{f}' \in \mathcal{U}_{i,i'} \setminus \hat{\mathcal{U}}$, $P \in \mathcal{P}_k(\boldsymbol{f})$ and $P' \in \mathcal{P}_k(\boldsymbol{f}')$, the alternating paths P and P' are either edge-disjoint or the same.*

For any subset $\mathcal{V} \subseteq \mathcal{U}_{i,i'} \setminus \hat{\mathcal{U}}$, let $\mathcal{P}_k(\mathcal{V})$ denote the set $\bigcup_{\boldsymbol{f} \in \mathcal{V}} \mathcal{P}_k(\boldsymbol{f})$ of all k-relevant alternating paths of the u-fans $\boldsymbol{f} \in \mathcal{V}$. It follows from Claim 8.3.1 that all of the alternating paths in $\mathcal{P}_k(\mathcal{V})$ are edge-disjoint. Thus, we can flip them all ***simultaneously*** to modify the types of the u-fans in

[2] If $P_v = P_w$, the set $\mathcal{P}_k(\boldsymbol{f})$ only contains one copy of this path, i.e. it is not a multiset.

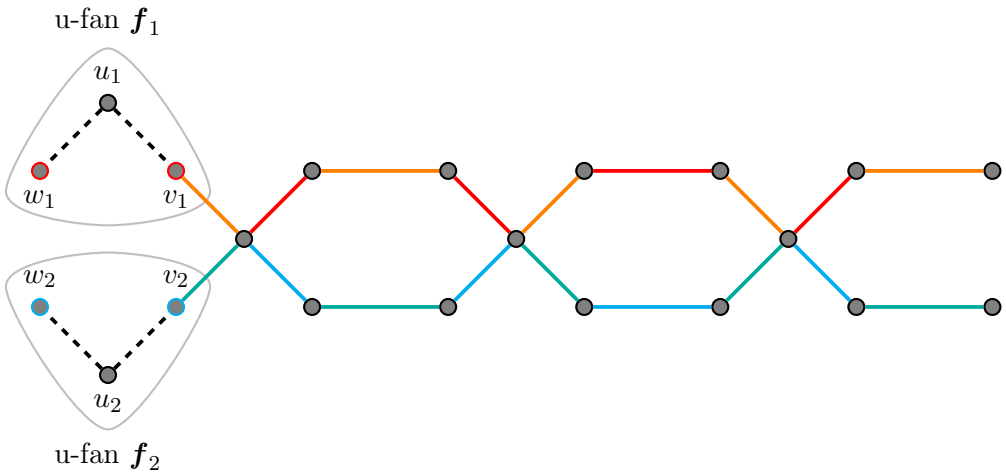

Figure 8.1: In this picture we have two different u-fans $\boldsymbol{f}_1 = (u_1, v_1, w_1, \cdot, C_i(j_1), C_i(j_1))$ and $\boldsymbol{f}_2 = (u_2, v_2, w_2, \cdot, C_i(j_2), C_i(j_2))$. When $j_1 \neq j_2$, we have $\{C_i(j_1), C_{2k}(j_1)\} \cap \{C_i(j_2), C_{2k}(j_2)\} = \emptyset$. Then two k-relevant alternating paths from v_1, v_2 of type-$\{C_i(j_1), C_{2k}(j_1)\}$ and type-$\{C_i(j_2), C_{2k}(j_2)\}$ are edge-disjoint.

$\mathcal{V}$ so that they are all contained in $C_{2k-1} \times C_{2k} \subseteq \mathcal{C}_k \times \mathcal{C}_k$, making them social.[3] See Figure 8.1 for an illustration.

We refer to a subset of u-fans $\mathcal{V} \subseteq \mathcal{U}$ such that $\mathcal{V} \subseteq \mathcal{U}_{i,i'}$ for some $i, i' \in [2\eta]$ as a **batch**. The subroutine Modify-Types modifies the types of a batch of u-fans $\mathcal{V} \subseteq \mathcal{U} \setminus \hat{\mathcal{U}}$, which we describe below in Algorithm 14.

Algorithm 14: Modify-Types$(\mathcal{V}, k)$

input: A batch $\mathcal{V} \subseteq \mathcal{U}_{i,i'} \setminus \hat{\mathcal{U}}$, for some $i, i' \in [2\eta]$

Let $\mathcal{P} \leftarrow \mathcal{P}_k(\mathcal{V})$ be the set of k-relevant alternating paths of the u-fans in $\mathcal{V}$

for $P \in \mathcal{P}$ **do**

  Call Flip-Path(P)

Good U-Fans. Whenever we modify the types of some subset of u-fans to make these u-fans social, we want to make sure that we are increasing the total number of social u-fans. Since applying Modify-Types$(\mathcal{V}, k)$ to some batch $\mathcal{V} \subseteq \mathcal{U}_{i,i'} \setminus \hat{\mathcal{U}}$ can change the types of up to $3|\mathcal{V}|$ u-fans that are not contained in $\mathcal{V}$—potentially damaging them—calling Modify-Types$(\mathcal{V}, k)$ for an arbitrary batch $\mathcal{V} \subseteq \mathcal{U}_{i,i'} \setminus \hat{\mathcal{U}}$ might be counterproductive.

To this end, we refer to a u-fan $\boldsymbol{f} \in \mathcal{U}$ as being **k-good** if it is not social and making a call to Modify-Types$(\{\boldsymbol{f}\}, k)$ does not change the type of any u-fan already in $\hat{\mathcal{U}}$. We say that a u-fan is **k-bad** if it is not k-good, and denote the set of k-bad u-fans by $\mathcal{B}_k \subseteq \mathcal{U}$. We can observe that calling Modify-Types$(\{\boldsymbol{f}\}, k)$ for a k-good u-fan increases the size of the set $\hat{\mathcal{U}}$ by at least 1.

[3] When we say that we can flip these paths *simultaneously*, we mean that we can flip them in any arbitrary order and still have the same effect on the coloring.

8.3.2 The Algorithm Sparsify-Types

The algorithm Sparsify-Types works in **iterations**. During each iteration, the algorithm finds a large batch $\mathcal{V}$ of good u-fans, modifies their types to make them social by making a call to Modify-Types, and then removes the damaged u-fans from $\mathcal{U}$. Once the number of social u-fans is at least $\lambda/100$, the algorithm terminates.

The pseudocode in Algorithm 15 gives a formal description of the algorithm. We note that, throughout the run of Algorithm 15, the separable collection $\mathcal{U}$ is updated during the calls to Modify-Types (where we change the missing colors assigned to u-fans) in Line 6 and when we remove damaged u-fans from $\mathcal{U}$ in Line 7. Whenever we refer to $\mathcal{U}$ or any subset of $\mathcal{U}$ (such as $\hat{\mathcal{U}}$, $\mathcal{U}_{i,i'}$ and $\mathcal{B}_k$) in Algorithm 15, these are defined with respect to the *current state* of the separable collection $\mathcal{U}$. In Section 8.3.4, we show how to compute these subsets efficiently.

Algorithm 15: Sparsify-Types($\mathcal{U}$)

1 Set $\mathcal{U} \leftarrow \{\boldsymbol{f} \in \mathcal{U} \mid \tau(\boldsymbol{f}) \in [q] \times [q]\}$
2 **while** $|\hat{\mathcal{U}}| < \lambda/100$ **do**
3 Let $k^\star \leftarrow \arg\min_{k\in[\eta]} |\mathcal{U}_{2k-1,2k} \cup \mathcal{U}_{2k-1,2k-1} \cup \mathcal{U}_{2k,2k}|$
4 Let $i^\star, i'^\star \leftarrow \arg\max_{1\le i<i'\le 2\eta} |\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}|$
5 Let $\mathcal{V} \leftarrow \mathcal{U}_{i^\star,i'^\star} \setminus \mathcal{B}_{k^\star}$
6 Call Modify-Types($\mathcal{V}, k^\star$)
7 Remove any damaged u-fans from $\mathcal{U}$
8 Set $\mathcal{U} \leftarrow \hat{\mathcal{U}}$

In Section 8.3.3, we analyze the algorithm Sparsify-Types and show that, after making a call to Sparsify-Types($\mathcal{U}$), the coloring χ and separable collection $\mathcal{U}$ satisfy the properties described in Lemma 8.2.1. In Section 8.3.4, we show how to implement the algorithm Sparsify-Types to run in time $O(m\eta^4 \log \mu)$.

8.3.3 Analysis of Sparsify-Types

We begin by showing that at least half of the u-fans in $\mathcal{U}$ initially have types in $[q] \times [q]$.

Claim 8.3.2. *By relabeling the colors (i.e. setting $\chi = \chi \circ \pi$ for a permutation $\pi : [\mu] \longrightarrow [\mu]$), we have that at least $3\lambda/5$ of the u-fans in $\mathcal{U}$ have types in $[q] \times [q]$.*

Proof. Given some u-fan $\boldsymbol{f} \in \mathcal{U}$ and $c \in [\mu]$, let $X_c^{\boldsymbol{f}}$ denote the indicator for the event that $c \in \tau(\boldsymbol{f})$, i.e. that the type of $\boldsymbol{f}$ contains the color c. Furthermore, let $X_c = \sum_{\boldsymbol{f}\in\mathcal{U}} X_c^{\boldsymbol{f}}$. Since the type of each (non-damaged) u-fan contains exactly 2 colors, we can see that $\sum_{c=1}^{\mu} X_c \le 2|\mathcal{U}| = 2\lambda$. By *relabeling the colors*, we can assume w.l.o.g. that $X_1 \ge \cdots \ge X_\mu$. It follows from this assumption that

$$\sum_{c=1}^{q} X_c \ge \frac{q}{\mu} \cdot \sum_{c=1}^{\mu} X_c.$$

Thus, we get that

$$\sum_{c=q+1}^{\mu} X_c \leq \left(1 - \frac{q}{\mu}\right) \cdot \sum_{c=1}^{\mu} X_c \leq 2\left(1 - \frac{q}{\mu}\right) \cdot \lambda \leq \frac{2}{5} \cdot \lambda.$$

Recalling that $\mu \geq 10\eta$, the last inequality follows from $q = 2\eta\lfloor \mu/2\eta \rfloor \geq 2\eta(\mu/2\eta - 1) \geq \mu - 2\eta \geq (4/5)\mu$. In other words, at most $2\lambda/5$ of the u-fans in $\mathcal{U}$ have types not in $[q] \times [q]$. □

The following lemma describes how the sizes of the sets $\mathcal{U}$ and $\hat{\mathcal{U}}$ change during each iteration of the algorithm.

Lemma 8.3.3. *Consider some iteration of Algorithm 15 where we call* Modify-Types$(\mathcal{V}, k^\star)$ *for a batch of u-fans $\mathcal{V}$. During this iteration, the size of $\hat{\mathcal{U}}$ increases by $|\mathcal{V}|$ and the size of $\mathcal{U}$ decreases by at most $3|\mathcal{V}|$.*

Proof. Since none of the u-fans in $\mathcal{V}$ are k-bad, we know that calling Modify-Types$(\{\boldsymbol{f}\}, k)$ for any $\boldsymbol{f} \in \mathcal{V}$ will not remove any u-fans from $\hat{\mathcal{U}}$ and will add the u-fan $\boldsymbol{f}$ to $\hat{\mathcal{U}}$ after changing its type. Thus, calling Modify-Types$(\mathcal{V}, k)$, which flips all of the k-relevant alternating paths corresponding to the u-fans in $\mathcal{V}$, adds each u-fan in $\mathcal{V}$ to $\hat{\mathcal{U}}$ without removing any u-fans from $\hat{\mathcal{U}}$. However, flipping the k-relevant alternating paths corresponding to a u-fan can damage up to 3 u-fans in $\mathcal{U}$. Thus, calling Modify-Types$(\mathcal{V}, k)$ can damage up to $3|\mathcal{V}|$ u-fans in $\mathcal{U}$, which are removed from $\mathcal{U}$ during Line 7. Thus, $|\hat{\mathcal{U}}|$ increases by $|\mathcal{V}|$ and $|\mathcal{U}|$ decreases by at most $3|\mathcal{V}|$. □

We can now prove the following claim, which lower bounds the size of $\mathcal{U}$ throughout the run of the algorithm.

Claim 8.3.4. *At the start of each iteration of Algorithm 15, we have that $|\mathcal{U}| \geq \lambda/2$.*

Proof. It follows from Claim 8.3.2 that Line 1 of Algorithm 15 only removes at most $2\lambda/5$ u-fans from $\mathcal{U}$. Thus, at the start of the first iteration, we have that that $|\mathcal{U}| \geq 3\lambda/5$. It follows from Lemma 8.3.3 that, if $|\hat{\mathcal{U}}|$ increases by Φ during some iteration, then $|\mathcal{U}|$ decreases by at most 3Φ during the same iteration. Thus, at the start of each iteration, we have that $|\mathcal{U}| \geq 3\lambda/5 - 3|\hat{\mathcal{U}}|$. Since the algorithm terminates once $|\hat{\mathcal{U}}| \geq \lambda/100$, it follows that $|\mathcal{U}| \geq 3\lambda/5 - 3\lambda/100 \geq \lambda/2$. □

The following lemma lower bounds the size of the batch of good edges $\mathcal{V}$ that we construct during each iteration and is the main technical component in the analysis of this algorithm.

Lemma 8.3.5. *During each iteration of Algorithm 15, we construct a batch $\mathcal{V} \subseteq \mathcal{U}_{i,i'}$ of k-good u-fans with size $|\mathcal{V}| \geq \lambda/(4\eta^2)$, for some $i, i' \in [2\eta]$ s.t. $i < i'$ and $k \in [\eta]$.*

Proof. Consider the state of the algorithm at the start of some iteration. We now show that the set $\mathcal{U}_{i^\star,i'^\star} \setminus \mathcal{B}_{k^\star}$ constructed during the iteration has size at least $\lambda/(4\eta^2)$.

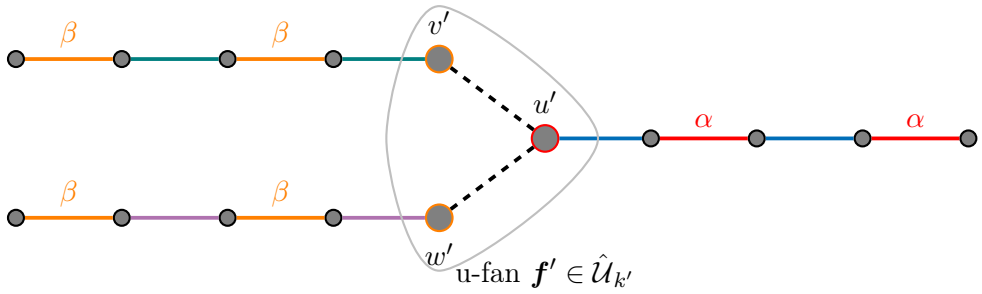

Figure 8.2: In this example, $\boldsymbol{f}' = (u', v', w', \alpha, \beta, \beta) \in \hat{\mathcal{U}}_{k'}$, where $k' \neq k$ and $\alpha = C_{2k'-1}(j_1), \beta = C_{2k'}(j_2)$. Then, for any $i \in \{2k-1, 2k\}$, u' could be the endpoint of a k-relevant path of type $\{\alpha = C_{2k'-1}(j_1), C_i(j_1)\}$, and v', w' could be endpoints of k-relevant paths of types $\{\beta = C_{2k'}(j_2), C_i(j_2)\}$. Thus, the total number of k-bad u-fans with respect to $\boldsymbol{f}'$ is at most 6.

We can observe that the subsets $\{\mathcal{U}_{i,i'}\}_{1 \leq i < i' \leq \eta} \cup \{\mathcal{U}_{i,i}\}_{i \in [\eta]}$ partition the set $\mathcal{U}$. Since, for any $i \in [2\eta]$, $\mathcal{U}_{i,i} \subseteq \hat{\mathcal{U}} \subseteq \mathcal{B}_{k^\star}$, it follows that

$$\mathcal{U} \setminus \mathcal{B}_{k^\star} = \bigcup_{1 \leq i < i' \leq \eta} (\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}). \tag{8.1}$$

By a simple averaging argument and Equation (8.1), we get that

$$|\mathcal{U}_{i^\star,i'^\star} \setminus \mathcal{B}_{k^\star}| \geq \frac{1}{\eta^2} \cdot \sum_{1 \leq i < i' \leq \eta} |\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}| = \frac{1}{\eta^2} \cdot |\mathcal{U} \setminus \mathcal{B}_{k^\star}|. \tag{8.2}$$

Now, for each $k \in [\eta]$, let $\hat{\mathcal{U}}_k := \mathcal{U}_{2k-1,2k} \cup \mathcal{U}_{2k-1,2k-1} \cup \mathcal{U}_{2k,2k}$. Then we have that $\hat{\mathcal{U}} = \bigcup_{k=1}^{\eta} \hat{\mathcal{U}}_k$. Since the sets $\{\hat{\mathcal{U}}_k\}_{k \in [\eta]}$ are all disjoint, it follows that $\sum_{k=1}^{\eta} |\hat{\mathcal{U}}_k| = |\hat{\mathcal{U}}| < \lambda/100$ and thus $|\hat{\mathcal{U}}_{k^\star}| \leq \lambda/(100\eta)$ by a simple averaging argument. We now prove the following claim, which upper bounds the number of k-bad u-fans in terms of $|\hat{\mathcal{U}}|$ and $|\hat{\mathcal{U}}_k|$.

Claim 8.3.6. *For each $k \in [\eta]$, we have that $|\mathcal{B}_k| \leq 7 \cdot |\hat{\mathcal{U}}| + 6\eta \cdot |\hat{\mathcal{U}}_k|$.*

Proof of claim. We first note that, for each non-social k-bad u-fan $\boldsymbol{f} \in \mathcal{B}_k$, there is some social u-fan $\boldsymbol{f}' \in \hat{\mathcal{U}}$ that is *responsible* for the u-fan $\boldsymbol{f}$ being k-bad, meaning that calling Modify-Types$(\{\boldsymbol{f}\}, k)$ changes the type of $\boldsymbol{f}'$. Thus, to upper bound the size of $\mathcal{B}_k$, it suffices to upper bound the number of non-social u-fans for which the social u-fans are responsible.

To this end, let $\boldsymbol{f}' \in \hat{\mathcal{U}}$. If $\boldsymbol{f}' \in \hat{\mathcal{U}} \setminus \hat{\mathcal{U}}_k$, then $\boldsymbol{f}'$ is responsible for at most 6 k-bad non-social u-fans. To see why, note that there are at most 2 k-relevant alternating paths ending at each vertex of $\boldsymbol{f}'$ (at most one with a color in C_{2k-1} and one with a color in C_{2k}) such that flipping these paths would remove $\boldsymbol{f}'$ from $\hat{\mathcal{U}}$, and for each of these alternating paths there is at most one non-social u-fan $\boldsymbol{f}$ such that calling Modify-Types$(\{\boldsymbol{f}\}, k)$ flips this path. More specifically, given a vertex $x \in \boldsymbol{f}'$ such that $c_{\boldsymbol{f}'}(x) = C_\ell(j)$, for some $\ell \notin \{2k-1, 2k\}$, the type of any such k-relevant alternating path ending at x is $\{C_\ell(j), C_i(j)\}$, for some $i \in \{2k-1, 2k\}\}$. See Figure 8.2 for an illustration.

On the other hand, if $\boldsymbol{f}' \in \hat{\mathcal{U}}_k$, then $\boldsymbol{f}'$ is responsible for at most 6η k-bad u-fans. To see why,

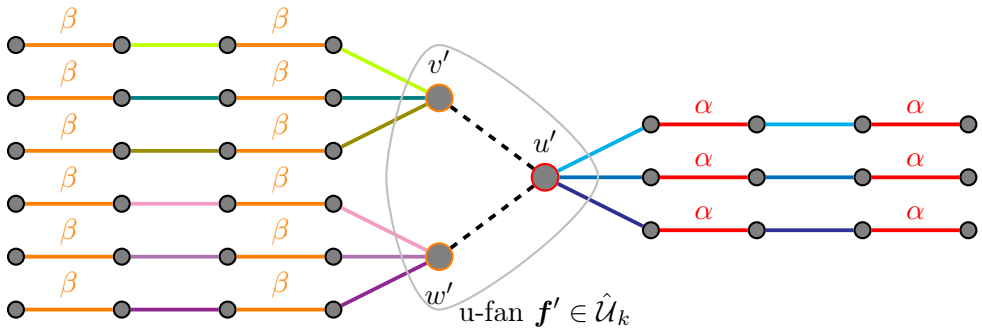

Figure 8.3: In this example, $\boldsymbol{f}' = (u', v', w', \alpha, \beta, \beta) \in \hat{\mathcal{U}}_k$, where $\alpha = C_{2k-1}(j_1), \beta = C_{2k}(j_2)$. Then, for any $i \in [2\eta] \setminus \{2k-1, 2k\}$, u' could be the endpoint of a k-relevant path of type $\{\alpha = C_{2k-1}(j_1), C_i(j_1)\}$, and v', w' could be endpoints of k-relevant paths of types $\{\beta = C_{2k}(j_2), C_i(j_2)\}$. Thus, the total number of k-bad u-fans with respect to $\boldsymbol{f}'$ is at most $6\eta - 6$.

note that there are at most $2\eta - 2$ k-relevant alternating paths ending at each vertex of $\boldsymbol{f}'$ (for each $i \in [2\eta] \setminus \{2k-1, 2k\}$, there is at most one such alternating path with a color in C_i) such that flipping these paths would remove $\boldsymbol{f}'$ from $\hat{\mathcal{U}}$, and for each of these alternating paths there is at most one non-social u-fan $\boldsymbol{f}$ such that calling Modify-Types$(\{\boldsymbol{f}\}, k)$ flips this path. More specifically, given a vertex $x \in \boldsymbol{f}'$ such that $c_{\boldsymbol{f}'}(x) = C_\ell(j)$, for some $\ell \in \{2k-1, 2k\}$, the type of any such k-relevant alternating path ending at x is $\{C_\ell(j), C_i(j)\}$, for some $i \in [2\eta] \setminus \{2k-1, 2k\}\}$. See Figure 8.3 for an illustration.

Thus, accounting for the social u-fans, the total number of k-bad u-fans is at most

$$6 \cdot |\hat{\mathcal{U}} \setminus \hat{\mathcal{U}}_k| + 6\eta \cdot |\hat{\mathcal{U}}_k| + |\hat{\mathcal{U}}| \leq 7 \cdot |\hat{\mathcal{U}}| + 6\eta \cdot |\hat{\mathcal{U}}_k|. \qquad \square$$

It follows from Claim 8.3.6 that $|\mathcal{B}_{k^\star}| \leq 7 \cdot \lambda/100 + 6\eta \cdot \lambda/(100\eta) \leq \lambda/4$. Since, by Claim 8.3.4, the size of $\mathcal{U}$ is at least $\lambda/2$, it follows from Equation (8.2) and the upper bound on $|\mathcal{B}_{k^\star}|$ that

$$|\mathcal{U}_{i^\star, i'^\star} \setminus \mathcal{B}_{k^\star}| \geq \frac{1}{\eta^2} \cdot |\mathcal{U} \setminus \mathcal{B}_{k^\star}| \geq \frac{1}{\eta^2} \cdot (|\mathcal{U}| - |\mathcal{B}_{k^\star}|) \geq \frac{1}{\eta^2} \cdot \left(\frac{\lambda}{2} - \frac{\lambda}{4}\right) \geq \frac{\lambda}{4\eta^2}. \qquad \square$$

Combining Lemmas 8.3.3 and 8.3.5, we get the following corollary, which upper bounds the number of iterations performed by the algorithm.

Corollary 8.3.7. *During each iteration of Algorithm 15, the size of $\hat{\mathcal{U}}$ increases by at least $\lambda/(4\eta^2)$.*

Thus, after $O(\eta^2)$ many iterations, we have that $|\hat{\mathcal{U}}| \geq \lambda/100$, so the algorithm sets $\mathcal{U} \leftarrow \hat{\mathcal{U}}$ and terminates. Since $\hat{\mathcal{U}}$ is a subset of a separable collection, it must be a separable collection itself. Furthermore, it follows from the definition of $\hat{\mathcal{U}}$ that the type of every u-fan in $\hat{\mathcal{U}}$ is contained in the subset of types $\bigcup_{k=1}^{\eta} \mathcal{C}_k \times \mathcal{C}_k$.

8.3.4 Implementation of Sparsify-Types

In this section, we show how to implement each iteration of Algorithm 15 in $O(m\eta^2 \log \mu)$ time. By the analysis in Section 8.3.3, the algorithm runs for $O(\eta^2)$ iterations. Hence, that would imply that the running time of Sparsify-Types is $O(m\eta^4 \log \mu)$.

We first note that our algorithm maintains the separable collection $\mathcal{U}$ explicitly. Thus, we can construct the separable collections $\{\hat{\mathcal{U}}_k\}_{k\in[\eta]}$ by initializing an empty separable collection (using our data structures) for each $\hat{\mathcal{U}}_k$ and then scanning through each of the u-fans in $\mathcal{U}$, adding them to the appropriate collection in $\{\hat{\mathcal{U}}_k\}_{k\in[\eta]}$ whenever they are contained in one. This can be done in $\eta \cdot O(m \log \mu)$ time. Given these separable collections, we can easily find $k^\star = \arg\min_{k\in[\eta]} |\hat{\mathcal{U}}_k|$. Similarly, given the set of $k^\star$-bad u-fans $\mathcal{B}_{k^\star}$, we can construct the separable collections $\{\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}\}_{1\le i<i'\le\eta}$ in $O(m\eta^2 \log \mu)$ time and then find the values $i^\star, i'^\star = \arg\max_{1\le i<i'\le\eta} |\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}|$.

We now show to compute the set of k-bad u-fans $\mathcal{B}_k$ for any $k \in [\eta]$ and implement the subroutine Modify-Types$(\mathcal{V}, k)$ in $O(m\eta \log \mu)$ time.

Finding k-Bad U-Fans. Following the proof of Claim 8.3.6, which upper bounds the number of k-bad u-fans, we note that finding all of the non-social k-bad u-fans can be done by simply traversing the k-relevant alternating paths starting at the vertices of the social u-fans. Specifically, for each social u-fan $\boldsymbol{f}' \in \hat{\mathcal{U}}$ and each vertex $x \in \boldsymbol{f}'$, one can traverse the k-relevant alternating paths with the color $c_{\boldsymbol{f}'}(x)$ starting at x. If $c_{\boldsymbol{f}'}(x) \in \mathcal{C}_k$, there are $2\eta - 2$ such paths. Otherwise, there are only 2. For each of these paths P, we can traverse the path in time $O(|P| \log \mu)$ using our data structures and find its other endpoint y. We can then check in $O(\log \mu)$ time using our data structures if there is a non-social u-fan $\boldsymbol{f}$ with a vertex at y such that calling Modify-Types$(\{\boldsymbol{f}\}, k)$ flips this path. If so, we add this u-fan $\boldsymbol{f}$ to the set of k-bad u-fans. After doing this for each social u-fan, we have found all of the non-social k-bad u-fans in $\mathcal{B}_k$.

For each $i \in [2\eta] \setminus \{2k, 2k-1\}$ and $\ell \in \{2k, 2k-1\}$, the total length of all $\{C_i(j), C_\ell(j)\}$-alternating paths, for all $j \in [r]$, is upper bounded by m, since all of these paths are edge disjoint. Thus, summing over all $i \in [2\eta] \setminus \{2k, 2k-1\}$ and $\ell \in \{2k, 2k-1\}$, it follows that the total length of all k-relevant alternating paths is $O(m\eta)$. Since we traverse each k-relevant alternating path at most twice during this process, this entire process can be implemented in $O(m\eta \log \mu)$ time using our data structures.

Implementing Modify-Types$(\mathcal{V}, k)$. We construct the set of k-relevant alternating paths $\mathcal{P}$ by scanning through each u-fan $\boldsymbol{f} \in \mathcal{V}$ and computing the k-relevant alternating paths corresponding to $\boldsymbol{f}$. By a similar argument as above, we conclude that the total time taken to compute and flip each of these paths is $O(\log \mu)$ times the total lengths of these paths, which is $O(m\eta)$. Thus, this can be done in $O(m\eta \log \mu)$ time.

8.4 The Algorithm Color-Small: Proof of Lemma 8.2.2

In this section, we describe and analyze the subroutine Color-Small, proving Lemma 8.2.2, which we restate below.

Lemma 8.2.2. *Given a graph G, a partial μ-coloring χ with colors C, and a separable collection $\mathcal{U}$ of size λ, the algorithm* Color-Small *extends the coloring χ to at least $\lambda/100$ uncolored edges. Furthermore, the algorithm is deterministic and runs in $O(m\mu^2 \log \mu)$ time.*

Proof. We begin by identifying the most common type $\tau^\star$ among the u-fans in $\mathcal{U}$. By a simple averaging argument, we know that $\ell \geq |\mathcal{U}|/\mu^2$ of the u-fans in $\mathcal{U}$ have type $\tau^\star$. We can find this type in $O(m \log \mu)$ time by scanning over each u-fan in $\mathcal{U}$ and counting the number of occurrences of each type. We then activate these u-fans by flipping only alternating paths with type $\tau^\star$, extending the coloring to ℓ uncolored edges in $O(n \log \mu)$ time, and remove the u-fans whose type changed during this process from $\mathcal{U}$. We repeat this process for $O(\mu^2)$ iterations, each time extending the coloring to at least a $1/\mu^2$ proportion of the u-fans in $\mathcal{U}$. Thus, after μ^2 iterations, we have that $|\mathcal{U}| \leq \lambda \cdot (1-1/\mu^2)^{\mu^2} \leq \lambda/2$. Since activating a u-fan can change the type of at most one other u-fan in the separable collection, it follows that we have extended the coloring to at least $\lambda/4 \geq \lambda/100$ edges. The total running time of this algorithm is $O(m\mu^2 \log \mu)$. □

8.5 The Algorithm Extend-Coloring: Proof of Lemma 8.2.3

In this section, we describe and analyze the subroutine Extend-Coloring, proving Lemma 8.2.3, which we restate below.

Lemma 8.2.3. *Given a graph G, a partial μ-coloring χ of G with colors C, a separable collection $\mathcal{U}$ of size λ, and an integer $\eta \geq 10$, the algorithm* Extend-Coloring *extends the coloring χ to at least $\lambda/100^{(\log \mu/\log \eta)+1}$ uncolored edges. Furthermore, the algorithm is deterministic and runs in time $O(m\eta^4 \log^2(\mu)/\log(\eta))$.*

8.5.1 The Algorithm Extend-Coloring

Let G be a graph, $\chi : E(G) \longrightarrow C \cup \{\perp\}$ be a partial μ-coloring of G, $\mathcal{U}$ a separable collection of size λ, and $\eta \geq 10$ an integer. The algorithm Extend-Coloring is recursive; given input $(G, \chi, \mathcal{U}, \eta)$, the algorithm does the following:

Base Case. If $\mu \leq 10\eta$, the algorithm calls Color-Small$(G, \chi, \mathcal{U})$ to extend the coloring χ to at least $\lambda/100$ uncolored edges.

Recursive Case. Otherwise, if $\mu > 10\eta$, the algorithm calls Sparsify-Types$(G, \chi, \mathcal{U}, \eta)$, which (by Lemma 8.2.1) modifies χ and $\mathcal{U}$ (without changing which edges are colored), and constructs disjoint subsets of colors $\mathcal{C}_1, \ldots, \mathcal{C}_\eta \subseteq C$ with the following properties:

1. For all $k \in [\eta]$, we have that $|\mathcal{C}_k| \leq \mu/\eta$.

2. The u-fans in $\mathcal{U}$ have types in $\bigcup_{k=1}^{\eta}(\mathcal{C}_k \times \mathcal{C}_k)$ and $|\mathcal{U}| \geq \lambda/100$.

The algorithm then uses the coloring χ and the subsets $\{\mathcal{C}_k\}_{k\in[\eta]}$ to compute the following subgraphs of G: For each $k \in [\eta]$, let $\mathcal{U}_k \subseteq \mathcal{U}$ be the subset of u-fans in $\mathcal{U}$ with a type in $\mathcal{C}_k \times \mathcal{C}_k$. We then define a set of edges $E_k := \chi^{-1}(\mathcal{C}_k) \cup E(\mathcal{U}_k)$, i.e. E_k is the set of all colored edges with colors in $\mathcal{C}_k$ and uncolored edges contained in the u-fans in $\mathcal{U}$ that have a type in $\mathcal{C}_k \times \mathcal{C}_k$, and let $G_k := (V, E_k)$ be the subgraph of G containing these edges. We emphasize that the subgraphs $\{G_k\}_{k\in[\eta]}$ are all edge-disjoint, which enables to proceed recursively on all of them 'in parallel' (without any possible conflicts). We define $\chi_k : E_k \longrightarrow \mathcal{C}_k \cup \{\perp\}$ to be the coloring of the graph G_k obtained by restricting the coloring χ to the edges in E_k. We also define $E_{\eta+1} := E(G) \setminus (E_1 \cup \cdots \cup E_\eta)$ and $\chi_{\eta+1} : E_{\eta+1} \longrightarrow (\mathcal{C} \setminus \bigcup_{i=1}^{\eta} \mathcal{C}_i) \cup \{\perp\}$ to be the coloring obtained by restricting the coloring χ to the edges in $E_{\eta+1}$.

For each $k \in [\eta]$, the algorithm calls Extend-Coloring$(G_k, \chi_k, \mathcal{U}_k, \eta)$, which modifies the coloring χ_k. Finally, we set χ to be the union of the colorings $\chi_1, \ldots, \chi_{\eta+1}$.

8.5.2 Analysis of Extend-Coloring

We prove Lemma 8.2.3 by induction on the value of

$$\texttt{depth}(\mu, \eta) := \max(\lceil \log_\eta(\mu/(10\eta)) \rceil, 0) \leq (\log \mu / \log \eta) + 1.$$

More specifically, we show that the algorithm extends the coloring χ to at least $\lambda/100^{\texttt{depth}(\mu,\eta)+1}$ edges in $O(m\eta^4 \log(\mu) \cdot (\texttt{depth}(\mu, \eta) + 1))$ time. Note that the value $\texttt{depth}(\mu, \eta)$ is an upper bound on the depth of the recursion tree of an instance: since the recursion bottoms when $\mu \leq 10\eta$, we have $\mu/\eta^d \leq 10\eta$, where d is the recursion depth.

We first note that, if $\mu \leq 10\eta$, the algorithm uses Color-Small to extend the coloring χ of G to at least $\lambda/100$ uncolored edges. It follows from Lemma 8.2.2 that the time it takes to compute this coloring is $O(m\mu^2 \log \eta) = O(m\eta^2 \log \mu)$. Furthermore, this is the base case of the induction and $\texttt{depth}(\mu, \eta) = 0$. Thus, Lemma 8.2.3 holds whenever $\mu \leq 10\eta$.

For the induction step, suppose that Lemma 8.2.3 holds whenever $\texttt{depth} \leq L$ for some integer $L \geq 0$ and that $\texttt{depth}(\mu, \eta) = L + 1$. In this case, the algorithm uses the subroutine Sparsify-Types to modify the coloring χ and the separable collection $\mathcal{U}$. Let $\tilde{\chi}$ and $\tilde{\mathcal{U}}$ denote the state of the coloring χ and the separable collection $\mathcal{U}$ after calling Sparsify-Types$(G, \chi, \mathcal{U}, \eta)$. The algorithm then splits the instance $(G, \tilde{\chi}, \tilde{\mathcal{U}}, \eta)$ into subproblems $\{(G_k, \tilde{\chi}_k, \tilde{\mathcal{U}}_k, \eta)\}_{k\in[\eta]}$ and recurses on each of these subproblems to modify the colorings $\{\tilde{\chi}_k\}_{k\in[\eta]}$. Let $\chi^\star_k$ denote the state of the coloring $\tilde{\chi}_k$ after calling Extend-Coloring$(G_k, \tilde{\chi}_k, \tilde{\mathcal{U}}_k, \eta)$, and let $\chi^\star_{\eta+1}$ denote the restriction of the coloring $\tilde{\chi}$ to the edges that are not contained in any G_k. The solution returned by the algorithm is the union of the colorings $\chi^\star_1, \ldots, \chi^\star_{\eta+1}$, which we denote by $\chi^\star$.

The following claim shows that these subproblems are all feasible.

Claim 8.5.1. *Each of the subproblems in $\{(G_k, \tilde{\chi}_k, \tilde{\mathcal{U}}_k, \eta)\}_{k\in[\eta]}$ is a valid instance. Moreover, the subgraphs corresponding to these η subproblems are edge-disjoint, and thus any color (re)assignments*

done for one subproblem have no effect on the other subproblems.

Proof. Since $\tilde{\chi}_k$ is the restriction of $\tilde{\chi}$ to G_k, it follows from the definition of G_k that $\tilde{\chi}_k$ is a coloring of G_k with colors $\mathcal{C}_k$. Since $\tilde{\mathcal{U}}_k \subseteq \tilde{\mathcal{U}}$ is a subset of a separable collection, it is a separable collection itself. Furthermore, the edges of each u-fan in $\tilde{\mathcal{U}}_k$ are contained in G_k, and for each u-fan $\boldsymbol{f} \in \tilde{\mathcal{U}}_k$, we have that $\tau(\boldsymbol{f}) \subseteq \mathcal{C}_k$. It follows immediately from the definition of the subgraphs G_k that they are edge-disjoint. □

Since $|\mathcal{C}_k| \leq \mu/\eta$ for each $k \in [\eta]$ and $\texttt{depth}(\mu/\eta, \eta) \leq \texttt{depth}(\mu, \eta) - 1 = L$, it follows by Claim 8.5.1 and the induction hypothesis that, for each $k \in [\eta]$, the coloring $\chi_k^\star$ has at least $|\tilde{\mathcal{U}}_k| \cdot 100^{-L}$ more colored edges than $\tilde{\chi}_k$. By Lemma 8.2.1, we know that $\tilde{\mathcal{U}} = \bigcup_{k=1}^{\eta} \tilde{\mathcal{U}}_k$. Since the domains of the colorings $\{\chi_k^\star\}_{k\in[\eta+1]}$ partition the edge set of G and the colors used by these colorings partition the set of colors C used by χ, it follows that the union of these colorings $\chi^\star$ is a partial coloring that assigns colors to at least

$$\sum_{k=1}^{\eta} |\tilde{\mathcal{U}}_k| \cdot 100^{-L} = |\tilde{\mathcal{U}}| \cdot 100^{-L} \geq |\mathcal{U}| \cdot 100^{-(L+1)}$$

more edges than χ, where the second inequality follows from the fact that we have $|\tilde{\mathcal{U}}| \geq |\mathcal{U}|/100$ by Lemma 8.2.1. Thus, the coloring $\chi^\star$ has the required properties.

To bound the running time of the algorithm, we can observe that, for each ℓ, the total running time at the ℓ^{th} level of the recursion tree is $O(m\eta^4 \log \mu)$. To see why this is the case, note that the subgraphs in each branch of the ℓ^{th} level of the recursion tree are edge-disjoint by Claim 8.5.1, and further note that the running time at any level other than the bottom level of the recursion is dominated by the time taken to handle the calls to Sparsify-Types. Given any such subgraph with m' edges, the time taken to handle the corresponding call to Sparsify-Types is $O(m'\eta^4 \log \mu)$ by Lemma 8.2.1. Due to the edge-disjointness of these subgraphs, summing over all these subgraphs yields a total running time of $O(m\eta^4 \log \mu)$ at this level. Similarly, the running time at the bottom level of the recursion is dominated by the time taken to handle calls to Color-Small, which by Lemma 8.2.2 gives rise to a runtime of $O(m\eta^2 \log \mu)$. Since the depth of the recursion tree is $\texttt{depth}(\mu, \eta) = O(\log \mu / \log \eta)$, the desired running time follows.

8.6 Implementation and Data Structures

In this section, we describe the key data structures that we use to implement an edge coloring χ and a separable collection $\mathcal{U}$, allowing us to efficiently implement the operations performed by our algorithms. Our data structures are almost identical to the data structures used by [ABB+25], with the modification that we use balanced binary trees instead of hashmaps to make our data structures deterministic.

We begin by describing the data structures and then show how they can be used to efficiently implement the queries described in Section 8.1.4.

Preprocessing the Graph. We can assume that, whenever we deal with a graph $G = (V, E)$ with n vertices and m edges, we have that $V = \{1, \ldots, n\}$, i.e. the vertices are the integers from 1 to n. We can achieve this by having an initial preprocessing phase where we sort the vertices in our graph G into a list $u_1, \ldots, u_n$ and replace each occurrence of u_i with i. This can easily be done in time $O(m \log n)$. Since the running time of our final algorithm (Theorem 1.1.2) is $\Omega(m \log n)$, the overhead of this preprocessing is negligible.

Implementing an Edge Coloring. Let $G = (V, E)$ be a graph of maximum degree Δ and let C be a set of μ colors. For each $u \in V$, we let $N(u)$ denote the edges in E incident on u. Let $C = \{c_1, \ldots c_\mu\}$ such that $c_1 \leq \cdots \leq c_\mu$ (recall that the colors used by our algorithm are always integers). Then we define $C[j] := \{c_{j'} \mid c_{j'} \leq j\}$. We implement an edge coloring $\chi : E \longrightarrow C$ of G using the following, for each $u \in V$:

- The map $\phi_u : N(u) \longrightarrow C$ where $\phi_u(e) := \chi(e)$ for all $e \in N(u)$.
- The map $\phi'_u : C \longrightarrow N(u)$ where $\phi'_u(c) := \{e \in N(u) \mid \chi(e) = c\}$.
- The set $\mathsf{miss}_\chi(u) \cap C[\deg_G(u) + 1]$.[4]

We implement all of the maps and sets using balanced binary trees, allowing us to perform membership queries, insertions and deletions in $O(\log \mu)$ time, since each of these maps and sets have size at most μ. Furthermore, we store an array of pointers to these binary trees for each $u \in V$, allowing us to retrieve any of these trees in $O(1)$ given the corresponding vertex u.

The map ϕ' allows us to check if a color $c \in C$ is available at a vertex $u \in V$ in $O(\log \mu)$ time, and if it is not, to find the edge $e \in N(u)$ with $\chi(e) = c$. Each time an edge e changes color under χ, we can easily update all of these data structures in $O(\log \mu)$ time. Furthermore, given $O(\log \mu)$ time query access to an edge coloring χ, we can initialize these data structures in $O(m \log \mu)$ time. We note that χ is a proper edge coloring if and only if $|\phi'_u(c)| \leq 1$ for all $u \in V$ and $c \in C$.

Since the binary trees used to implement the maps ϕ_u stores m elements, it follows that it can be implemented with $O(m)$ space. Similarly, the maps ϕ'_u store $2m$ elements in total (if $\{e \in N(u) \mid \chi(e) = c\} = \emptyset$, then we do not store anything for $\phi'_u(c)$) and thus can be implemented with $O(m)$ space since each element has size $O(1)$ (recall that $|\phi'_u(c)| \leq 1$ since the coloring is proper). Since $\sum_u |\mathsf{miss}_\chi(u) \cap C[\deg_G(u) + 1]| = O(m)$, the sets $\mathsf{miss}_\chi(u) \cap C[\deg_G(u) + 1]$ can be implemented in $O(m)$ space.

Implementing a Separable Collection. We implement a separable collection $\mathcal{U}$ in a similar manner using the following, for each $u \in V$:

- The map $\psi_u : C \longrightarrow \mathcal{U}$ where $\psi_u(c) := \{\boldsymbol{f} \in \mathcal{U} \mid u \in \boldsymbol{f}, c_{\boldsymbol{f}}(u) = c\}$.
- The set $C_{\mathcal{U}}(u) := \{c_{\boldsymbol{f}}(u) \mid \boldsymbol{f} \in \mathcal{U}, u \in \boldsymbol{f}\}$.

[4] We take this intersection with $C[\deg_G(u) + 1]$ instead of maintaining $\mathsf{miss}_\chi(u)$ directly to ensure that the space complexity and initialization time of the data structures are $\tilde{O}(m)$ and not $\Omega(\Delta n)$.

- The set $\overline{C}_{\mathcal{U}}(u) := (\mathsf{miss}_{\chi}(u) \cap C[\deg_G(u)+1]) \setminus C_{\mathcal{U}}(u)$.

We again implement all of the maps and sets using balanced binary trees, allowing us to access and change entries in $O(\log \mu)$ time. We note that, since $\mathcal{U}$ is separable, $|\psi_u(c)| \leq 1$ for all $u \in V$ and $c \in C$. Each time we remove a color $c \in C$ from the palette $\mathsf{miss}_{\chi}(u)$ of a vertex $u \in V$, we can update $\overline{C}_{\mathcal{U}}(u)$ in $O(\log \mu)$ time and check $\psi_u(c)$ in $O(\log \mu)$ time to find any u-component that has been damaged. Each time we add or remove a u-fan from $\mathcal{U}$, we can update all of these data structures in $O(\log \mu)$ time. Furthermore, we can initialize these data structures for an empty collection in $O(m \log \mu)$ time by creating a empty maps ψ_u and empty sets $C_{\mathcal{U}}(u)$ for each $u \in V$ and copying the sets $\overline{C}_{\mathcal{U}}(u) = \mathsf{miss}_{\chi}(u) \cap C[\deg_G(u)+1]$ for each $u \in V$ which are maintained by the data structures for the edge coloring χ. Since $\mathcal{U}$ is separable, we can see that $\overline{C}_{\mathcal{U}}(u) \neq \emptyset$. Thus, whenever we want a color from the set $\mathsf{miss}_{\chi}(u) \setminus C_{\mathcal{U}}(u)$, it suffices to take an arbitrary color from the set $\overline{C}_{\mathcal{U}}(u)$.

For each $\boldsymbol{f} \in \mathcal{U}$, we can see that $\boldsymbol{f}$ is contained at most 3 different ψ_u. Thus, the total space required to store the binary trees that implement the ψ_u is $O(|\mathcal{U}|)$. Since $\mathcal{U}$ is separable, the u-fans in $\mathcal{U}$ are edge-disjoint, and thus $|\mathcal{U}| \leq m$. It follows that the maps ψ_u can be stored with $O(m)$ space. For each $u \in V$, we can observe that $|C_{\mathcal{U}}(u)| \leq \deg_G(u)$ since at most $\deg_G(u)$ many u-fans in $\mathcal{U}$ contain the vertex u, and $|\overline{C}_{\mathcal{U}}(u)| \leq \deg_G(u)+1$ since $\overline{C}_{\mathcal{U}}(u) \subseteq C[\deg_G(u)+1]$. Thus, the total space required to store the sets $\{C_{\mathcal{U}}(u)\}_{u \in V}$ and $\{\overline{C}_{\mathcal{U}}(u)\}_{u \in V}$ is $O(m)$.

8.6.1 Implementing the Operations from Section 8.1.4

We now describe how to implement each of the operations from Section 8.1.4.

Implementing INITIALIZE(G, χ)**.** Suppose that we are given the graph G and $O(\log \mu)$ time query access to an edge coloring χ of G. We can initialize the data structures used to maintain the maps ϕ_u and ϕ'_u in $O(m \log \mu)$ time. We can then scan through the vertices $u \in V$ and initialize the sets $\mathsf{miss}_{\chi}(u) \cap C[\deg_G(u)+1]$ in $O(m \log \mu)$ time. We can then initialize the data structures for an empty separable collection in $O(m \log \mu)$ time by creating empty maps ψ_u and initializing the sets $C_{\mathcal{U}}(u) \leftarrow \emptyset$ and $\overline{C}_{\mathcal{U}}(u) \leftarrow \mathsf{miss}_{\chi}(u) \cap C[\deg_G(u)+1]$ for each $u \in V$.

Implementing INSERT$_{\mathcal{U}}(\boldsymbol{f})$**.** By performing at most 3 queries to the maps ψ_u, we can check if $\mathcal{U} \cup \{\boldsymbol{f}\}$ is separable. If so we can update the ψ_u and the sets $C_{\mathcal{U}}(x)$ and $\overline{C}_{\mathcal{U}}(x)$ for $x \in \boldsymbol{f}$ in $O(\log \mu)$ time in order to insert $\boldsymbol{f}$ into $\mathcal{U}$. Otherwise, we return `fail`.

Implementing DELETE$_{\mathcal{U}}(\boldsymbol{f})$**.** We can first make a query to the ψ_u to ensure that $\boldsymbol{f} \in \mathcal{U}$. If so, we can update the ψ_u and the sets $C_{\mathcal{U}}(x)$ and $\overline{C}_{\mathcal{U}}(x)$ for $x \in \boldsymbol{f}$ in $O(\log \mu)$ time to remove $\boldsymbol{f}$ from $\mathcal{U}$.

Implementing FIND-U-FAN$_{\mathcal{U}}(x, c)$**.** We make a query to ψ_x and check if there is an element $\psi_x(c)$. If no such element is contained in ψ_x, then return `fail`. Otherwise, return the unique u-fan in the set $\psi_x(c)$. This takes $O(\log \mu)$ time.

Implementing MISSING-COLOR$_{\mathcal{U}}(x)$**.** Return an arbitrary color from the set $\overline{C}_{\mathcal{U}}(x)$.

8.7 Extension to Vizing's Theorem for Multigraphs

The deterministic algorithm presented in this chapter extends naturally to multigraphs, proving Theorem 1.1.4, which we restate below.

Theorem 1.1.4. *There is a deterministic algorithm that, given a multigraph G with maximum degree Δ and maximum multiplicity μ, computes a $(\Delta+\mu)$-coloring of G in $m \cdot 2^{O(\sqrt{\log \Delta})} \cdot \log n = m^{1+o(1)}$ time.*

Extending this result to multigraphs follows the exact same approach as Section 7.7. In particular, we must replace the definition of a separable collection with the generalization to multigraphs (see Section 7.7.1) and instead of using Lemma 8.1.3 to construct a separable collection of u-fans, we use Lemma 7.7.7, which is a generalization of this lemma for multigraphs.

Crucially, in the same way as Section 7.7, the subroutines in our algorithm that operate directly on separable collections extend seamlessly to multigraphs. Analogously to Lemma 7.7.8, the main subroutine Extend-Coloring in our deterministic algorithm extends directly to multigraphs; Lemma 8.2.3 holds as is for mulitgraphs with an almost identical proof.

Proof of Theorem 1.1.4. Following the same approach as Section 7.7.2, we can extend the proof from Section 8.2.1 to Theorem 1.1.4 by using the generalizations of Lemma 8.1.3 and Lemma 8.2.3 to multigraphs (which are essentially identical) and slightly tweaking the Euler partitioning to deal with a $(\Delta+\mu)$-coloring instead of a $(\Delta+1)$-coloring.

Part III

Fast Dynamic Edge Coloring

Chapter 9

Dynamic Edge Coloring I: Shorter Multi-Step Vizing Chains

In this chapter, we give our full multi-step Vizing chain algorithm for extending a $(\Delta+1)$-coloring, proving Theorem 1.1.6, which we restate below.

Theorem 1.1.6. *There is an algorithm that, given a dynamic graph G with maximum degree at most Δ, maintains a $(\Delta+1)$-coloring of G with $\tilde{O}(\Delta^2+\sqrt{\Delta n})$ worst-case update time, against an adaptive adversary, with high probability.*

Notation. This chapter uses the basic notation described in Section 3.1. We introduce the rest of the notation used throughout this chapter in Section 9.1. We note that, for convenience, the notation used for Vizing fans and chains throughout this chapter is different from the notation in Section 3.2. Additionally, this chapter uses slightly different notation than the sketch of this proof in Section 5.2.

9.1 Preliminaries

We now describe the main components used in our multi-step Vizing chain construction. We note that we use different notation to Section 3.2 for convenience.

The Algorithm VizingFan

The first component of our algorithm is a standard Vizing fan construction, but with one simple modification. The algorithm VizingFan takes as input some uncolored edge (u,v) and colors $c_u \in \mathsf{miss}_\chi(u)$ and $c_v \in \mathsf{miss}_\chi(v)$. It then proceeds to either extend χ to the uncolored edge (u,v) by shifting colors around u, or constructs a Vizing fan around the vertex u while ensuring that (1) none of the vertices v_i participating in the fan have $c_u \in \mathsf{miss}_\chi(v_i)$, and (2) that the color of the first vertex in the fan is not c_v. Algorithm 16 gives a formal description of this procedure.

Algorithm 16: VizingFan(χ, u, v, c_u, c_v)

$i \leftarrow 0$ and $v_0 \leftarrow v$
Let $c_0 \in \mathsf{miss}_\chi(v_0) \setminus \{c_v\}$
while $c_i \notin \{c_0, \ldots, c_{i-1}\}$ and $c_i \notin \mathsf{miss}_\chi(u)$ **do**
    Let (u, v_{i+1}) be the edge with color $\chi(u, v_{i+1}) = c_i$
    Let $c_{i+1} \in \mathsf{miss}_\chi(v_{i+1})$
    if $c_u \in \mathsf{miss}_\chi(v_{i+1})$ **then**
        $c_{i+1} \leftarrow c_u$
    $i \leftarrow i + 1$
if $c_i \in \mathsf{miss}_\chi(u)$ **then**
    for $j = 0 \ldots i$ **do**
        $\chi(u, v_j) \leftarrow c_j$
    return χ
else
    return $(v_0, c_0), \ldots, (v_i, c_i)$

We refer to the sequence $(v_0, c_0), \ldots, (v_i, c_i)$ returned by Algorithm 16 as a Vizing fan. We say that a Vizing fan is c-primed if $c = c_i$.

The Algorithm Vizing

We now give an algorithm Vizing which builds a Vizing chain using VizingFan as a subroutine and extends the coloring χ to the uncolored edge (u, v). The algorithm also takes colors $c_u \in \mathsf{miss}_\chi(u)$ and $c_v \in \mathsf{miss}_\chi(v)$ as input. We make some small modifications to the standard implementation of this algorithm for technical reasons that will become clear later on. Algorithm 17 gives a formal description of this procedure.

Algorithm 17: Vizing(χ, u, v, c_u, c_v)

if VizingFan(χ, u, v, c_u, c_v) extends χ **then**
    $\chi \leftarrow$ VizingFan(χ, u, v, c_u, c_v)
    return χ
$\mathbf{F} = (v_0, c_0), \ldots, (v_k, c_k) \leftarrow$ VizingFan(χ, u, v, c_u, c_v)
if $\mathbf{F}$ is *not* c_v-primed **then**
    Let $c \in \mathsf{miss}_\chi(u) \setminus \{c_u\}$
    Let P denote the maximal $\{c, c_k\}$-alternating path starting at u
if $\mathbf{F}$ is c_v-primed **then**
    Let c_i be the first appearance of c_v in $c_0, \ldots, c_k$
    Let P denote the maximal $\{c_u, c_v\}$-alternating path starting at v_i
Extend χ to (u, v) by flipping the path P and shifting colors around u (see Lemma 9.1.1)
return χ

Thus, running $\mathsf{Vizing}(\chi, u, v, c_u, c_v)$ extends the coloring χ to the uncolored edge (u, v) by shifting colors around the vertex u and flipping the colors of the alternating path P. The analysis of the case where $\mathbf{F}$ is not c_v-primed is the same as in the standard Vizing chain construction [Sin19]. The case where $\mathbf{F}$ is c_v-primed follows from a similar argument. The following lemma summarises the behaviour of this algorithm.

Lemma 9.1.1. *Algorithm 17 extends the coloring χ to the edge (u, v) in time $\tilde{O}(\Delta + |P|)$.*

Proof. We begin by showing that the path P is well-defined. In the case that $\mathbf{F}$ is not c_v-primed, we know that $c \in \mathsf{miss}_\chi(u)$ and $c_k \notin \mathsf{miss}_\chi(u)$, so there is a $\{c, c_k\}$-alternating path starting at u. In the case that $\mathbf{F}$ is c_v-primed, we know by the properties of our Vizing fan construction that $c_v \in \mathsf{miss}_\chi(v_i)$ and $c_u \notin \mathsf{miss}_\chi(v_i)$, so there is a $\{c_u, c_v\}$-alternating path starting at u. Thus, in both cases, the path P is well-defined. We now describe how to extend the coloring χ to (u, v) in both cases.

If $\mathbf{F}$ is not c_v-primed: Let c_j be the first appearance of c_k. We consider the cases where the P does or does not have v_j as an endpoint. If P does not end at v_j, then we can shift the colors of the first $j + 1$ edges in the fan by setting $\chi(u, v_0) \leftarrow c_0, \ldots, \chi(u, v_j) \leftarrow c_j$ and flip the colors of the alternating path P. If P does end at v_j, then we flip the colors of the alternating path P, shift the colors of the fan by setting $\chi(u, v_0) \leftarrow \chi(u, v_1), \ldots, \chi(u, v_{k-1}) \leftarrow \chi(u, v_k)$, and set $\chi(u, v_k) \leftarrow c_k$. Note that, while shifting the fan, we have $\chi(u, v_{i+1}) = c \neq c_k$.

If $\mathbf{F}$ is c_v-primed: We consider the cases where the P does or does not have u as an endpoint. If P does not end at u, then the edge (u, v_{i+1}) (which has color c_v) is not contained in P. Thus, we can shift the colors of the first i edges in the fan by setting $\chi(u, v_0) \leftarrow c_0, \ldots, \chi(u, v_{i-1}) \leftarrow c_{i-1}$, flip the colors of the alternating path P, and set $\chi(u, v_i) \leftarrow c_u$. If P does end at u, then the edge (u, v_{i+1}) is contained in P. Thus, we can flip the colors of the alternating path P, shift the colors of the fan by setting $\chi(u, v_0) \leftarrow \chi(u, v_1), \ldots, \chi(u, v_{k-1}) \leftarrow \chi(u, v_k)$, and set $\chi(u, v_k) \leftarrow c_v$. Note that, while shifting the fan, we have $\chi(u, v_{i+1}) = c_u \neq c_i$.

Using standard data structures, we can implement this algorithm to run in time $\tilde{O}(\Delta + |P|)$. □

The following lemma summarises the key properties of the path P considered by Algorithm 17.

Lemma 9.1.2. *The path P considered by the algorithm satisfies one of the following properties:*

1. *P is a maximal $\{c', c''\}$-alternating path in χ starting at u where $\{c', c''\} \cap \{c_u, c_v\} = \emptyset$.*
2. *P is a maximal $\{c_u, c_v\}$-alternating path in χ starting at neither u nor v.*[1]

These properties of the path P will be crucial in our analysis later on. We refer to an alternating path satisfying Condition 1 (resp. Condition 2) above as *non-overlapping* (resp. *overlapping*).

[1] Note that, in this case, the path P starts at the vertex v_i of the fan such that c_i is the first appearance of c_v in $c_0, \ldots, c_k$. Since $c_0 \neq c_v$, we know that $v_i \neq v$.

The Algorithm TruncatedVizing

We now give an algorithm TruncatedVizing, which takes the same input as Vizing along with an additional argument $t \in \mathbb{N}$. It then proceeds to extend the coloring χ to the edge e in the same way as Vizing, with one difference: If the maximal alternating path P that is flipped by Vizing has length greater than t, the algorithm only flips the first $t-1$ edges of P and leaves the t^{th} edge in the path P uncolored. It then returns the new coloring obtained after applying this procedure along with the edge left uncolored (as long as P had length greater than t). Algorithm 18 gives a formal description of this procedure.

Algorithm 18: TruncatedVizing$(\chi, u, v, c_u, c_v, t)$

Let $P = e_1, \ldots, e_t$ denote the alternating path considered by Vizing(χ, u, v, c_u, c_v)
if $|P| \leq t$ **then**
  $\chi \leftarrow$ Vizing(χ, u, v, c_u, c_v)
  return χ
if P is non-overlapping **then**
  Let (u, v_i) be the first edge in P
  for $j = 0 \ldots i-1$ **do**
    $\chi(u, v_j) \leftarrow c_j$
if P is overlapping **then**
  Let v_i be the first vertex in P
  for $j = 0 \ldots i-1$ **do**
    $\chi(u, v_j) \leftarrow c_j$
  $\chi(u, v_i) \leftarrow c_u$
Flip the colors of the first $t-1$ edges in the $\{c', c''\}$-alternating path P
$\chi(e_t) \leftarrow \perp$
$(u', v') \leftarrow e_t$
Let $c'_u \in \mathsf{miss}_\chi(u') \cap \{c', c''\}$ and $c'_v \in \mathsf{miss}_\chi(v') \cap \{c', c''\}$
return χ, u', v', c'_u, c'_v

9.2 Our Multi-Step Vizing Chain Algorithm

In this section, we prove the following theorem, which implies Theorem 1.1.6.

Theorem 9.2.1. *Given a undirected simple graph $G = (V, E)$ on n vertices and maximum degree Δ, as well as and a partial $(\Delta+1)$-edge coloring χ of G with an uncolored edge e, we can extend χ to the edge e in $\tilde{O}(\Delta^2 + \sqrt{\Delta n})$ expected time.*

We define parameters $\ell := 10^2 \log n$ and $L := 10^3 \ell^2 (\Delta^2 + \sqrt{\Delta n})$. Our algorithm is given an uncolored edge (u, v) and starts by attempting to construct a Vizing chain by calling Vizing. If the Vizing chain has length $\Omega(L)$, then our algorithm instead calls Truncated-Vizing and randomly

truncates the Vizing chain after $O(L)$ steps. It then repeats this process, moving the uncolored edge around the graph by randomly truncating long Vizing chains, until it finds a short Vizing chain and successfully extends the coloring. Lemma 9.2.2 gives a formal description of this procedure.

Algorithm 19: ExtendColoring(χ, u, v)

```
Let c_u ∈ miss_χ(u) and c_v ∈ miss_χ(v)
repeat
    Let P = u_1, ..., u_{k+1} be the alternating path considered by Vizing(χ, u, v, c_u, c_v)
    if k ≤ 2L + 2 then
        χ ← Vizing(χ, u, v, c_u, c_v)
        return χ
    Sample i ∼ [L] independently and u.a.r.
    (χ, u, v, c_u, c_v) ← TruncatedVizing(χ, u, v, c_u, c_v, i + 1)
```

Using standard data structures, we can implement each iteration of Algorithm 19 to run in time $\tilde{O}(L) \leq \tilde{O}(\Delta^2 + \sqrt{\Delta n})$. The following lemma, which we prove in Section 9.2.1, shows that the algorithm terminates after $\tilde{O}(1)$ many iterations with constant probability.

Lemma 9.2.2. *Algorithm 19 terminates within $\ell+1$ iterations with probability at least $1/(160 \log n)$.*

9.2.1 Analysis

In order to analyse our algorithm, we define a (possibly infinite) rooted meta-tree $\mathcal{T}$ which captures information about all of the different possible executions of our algorithm.

The Meta-Tree $\mathcal{T}$

The meta-tree $\mathcal{T}$ is rooted at a meta-node r. Every meta-node $x \in \mathcal{T}$ corresponds to an iteration of Algorithm 19, and the root-to-x path from r to x in $\mathcal{T}$ corresponds to the execution of Algorithm 19 leading to this iteration. For each such meta-node x, we denote the state of an object Φ in Algorithm 19 at the start of the corresponding iteration of the algorithm by $\Phi^{(x)}$, e.g. $\chi^{(x)}$ denotes the coloring χ at the start of the corresponding iteration of the algorithm. Given a meta-node x, $P^{(x)}$ denotes the alternating path considered during the corresponding iteration of the algorithm. Consider the following definition:

- We say that x is *terminal* if the length of $P^{(x)}$ is at most $2L + 2$.

Each meta-node x has no children in $\mathcal{T}$ if it is terminal. Otherwise, x has exactly L children in $\mathcal{T}$, each one corresponding to a different choice of edge in $P^{(x)}$ for truncating the alternating path.[2]

Random Walks in $\mathcal{T}$. We first note that there is a one-to-one correspondence between root-to-leaf paths in $\mathcal{T}$ and executions of Algorithm 19. Furthermore, an execution of our algorithm defines a

[2]Intuitively, one can visualise constructing $\mathcal{T}$ by starting at the root r, where $\chi^{(r)} = \chi$ and $(u^{(r)}, v^{(r)}) = (u, v)$, and considering all possible executions of Algorithm 19 for the different random choices made by the algorithm.

random walk on $\mathcal{T}$ that starts at the root and, at each step, independently and uniformly samples a random child of the current meta-node (as long as the current meta-node is not terminal). Our algorithm successfully extends the coloring χ if and only if this random walk encounters a terminal meta-node. To this end, we show that such a random walk encounters a terminal meta-node within $O(\log n)$ steps with probability $\Omega(1/\log n)$, proving Lemma 9.2.2.

Classifications of Meta-Nodes. Given a meta-node $x \in \mathcal{T}$, let $\tau^{(x)}$ denote the colors of the alternating path $P^{(x)}$ that we consider during this iteration. We refer to any pair of colors as a *type*, and to $\tau^{(x)}$ as the type of x. Let $\tilde{P}^{(x)}$ denote the alternating path obtained by truncating the $\tau^{(x)}$-alternating path $P^{(x)}$ after the first $L+1$ edges.[3] Let $r = x_0, \ldots, x_i = x$ denote the root-to-x path in $\mathcal{T}$. Consider the following definitions:

- We say that x is *α-dirty* if $\tau^{(x)} \cap (\tau^{(x_0)} \cup \cdots \cup \tau^{(x_{i-2})}) \neq \emptyset$ and $\tau^{(x)} \cap \tau^{(x_{i-1})} = \emptyset$.
- We say that x is *β-dirty* if $\tau^{(x)} = \tau^{(x_{i-1})}$.
- We say that x is *α-contaminated* (resp. *β-contaminated*) if it at least a $1/(10\ell)$ proportion of its children are *α-dirty* (resp. *β-dirty*).
- We say that x is *damaged* if it is **not** dirty and $P^{(x)}$ is **not** a maximal $\tau^{(x)}$-alternating path in $\chi^{(r)}$ such that $\chi^{(r)}(e) = \chi^{(x)}(e)$ for all $e \in P^{(x)}$.

Note that a meta-node x cannot be both α-dirty and β-dirty, but it can be both α-contaminated and β-contaminated. We say that a meta-node x is dirty (resp. contaminated) if it is α-dirty or β-dirty (resp. α-contaminated or β-contaminated). Furthermore, it follows from Lemma 9.1.2 that either $\tau^{(x)} = \tau^{(x_{i-1})}$ or $\tau^{(x)} \cap \tau^{(x_{i-1})} = \emptyset$, so x is not dirty if and only if $\tau^{(x)} \cap (\tau^{(x_0)} \cup \cdots \cup \tau^{(x_{i-1})}) = \emptyset$.

Properties of the Meta-Tree $\mathcal{T}$

We now describe some properties of the meta-tree $\mathcal{T}$ that will be useful while analyzing the behaviour of random walks in $\mathcal{T}$. We begin with the following lemmas which show that contaminated meta-nodes have lots of terminal children.

Lemma 9.2.3. *Let x be an α-contaminated meta-node within the first ℓ levels of $\mathcal{T}$. Then at least a $1/(40\ell)$ proportion of the children of x are terminal.*

Proof. Let $r = x_0, \ldots, x_i = x$ denote the root-to-x path in $\mathcal{T}$. Since x is α-contaminated, it has at least $L/(10\ell)$ α-dirty children $y_1, \ldots, y_k$. Since $i \leq \ell$, at most 2ℓ colors appear in $\tau^{(x_0)} \cup \cdots \cup \tau^{(x_{i-2})}$. Thus, each of the α-dirty children $y_1, \ldots, y_k$ has one of at most $2\ell(\Delta+1) \leq 4\ell\Delta$ many types. Since $\tau^{(x)}$ is disjoint from the types of the α-dirty children of x, we have that the alternating paths $P^{(y_1)}, \ldots, P^{(y_k)}$ corresponding to the children of x are all maximal alternating paths with the same colors in $\chi^{(y_1)}$ and thus have a total length of at most $4\ell\Delta n$ (since the total length of the maximal

[3] Note that the alternating path $\tilde{P}^{(x)}$ is *not* maximal, while the alternating path $P^{(x)}$ *is* maximal.

alternating paths of any type is at most n). Furthermore, each alternating path in the collection $P^{(y_1)}, \ldots, P^{(y_k)}$ appears at most twice (once in each orientation), so we have at least $L/(20\ell)$ distinct alternating paths. Thus, by an averaging argument, at least half of these paths have length at most $8\ell\Delta n \cdot (20\ell/L) \leq L$, and hence correspond to terminal meta-nodes. □

Lemma 9.2.4. *Let x be a β-contaminated meta-node within the first ℓ levels of $\mathcal{T}$. Then at least a $1/(20\ell)$ proportion of the children of x are terminal.*

Proof. Since x is β-contaminated, it has at least $L/(10\ell)$ β-dirty children $y_1, \ldots, y_k$. Let τ denote the type of x (which is the same as the types of $y_1, \ldots, y_k$). Consider the coloring $\chi^{(y_1)}$ and the uncolored edge $(u^{(y_1)}, v^{(y_1)})$. Let P' (resp. P'') denote the τ-alternating path starting at $u^{(y_1)}$ (resp. $v^{(y_1)}$), and let w' (resp. w'') denote it's other endpoint. Let Q denote the path or cycle obtained by concatenating P', $(u^{(y_1)}, v^{(y_1)})$, and P''. Then the colorings $\chi^{(y_1)}, \ldots, \chi^{(y_k)}$ only differ on the colors of the edges in Q. Furthermore, in each of these colorings, Q always consists of either 1 or 2 τ-alternating paths and an uncolored edge.

Now, consider some meta-node y_i and it's corresponding vertex $u^{(y_i)}$ in Q. There is an edge $f^{(y_i)}$ connecting $u^{(y_i)}$ to the first vertex on the τ-alternating path $P^{(y_i)}$. We either have that the other endpoint of $f^{(y_i)}$ is w' or w'', or that $P^{(y_i)}$ is vertex disjoint from Q. In the former case, we say that y_i is a *bad extension point*. Since there are only 2Δ many edges incident on w' and w'', there are at most 2Δ bad extension points. Let $\{z_1, \ldots, z_{k'}\} \subseteq \{y_1, \ldots, y_k\}$ be the subset of the y_i that are not bad extension points. Now, consider the collection of τ-alternating paths $P^{(z_1)}, \ldots, P^{(z_{k'})}$. Since these are all vertex disjoint from Q, they are all maximal τ-alternating paths with the same colors in $\chi^{(y_1)}$, and hence have a total length of at most n without counting repeated paths. Furthermore, each alternating path in the collection $P^{(z_1)}, \ldots, P^{(z_{k'})}$ appears at most 2Δ times since each endpoint of each path has maximum degree Δ, and thus can only be incident on at most Δ edges $f^{(y_i)}$ connecting the path to some $u^{(y_i)}$. Thus, their total length (counting repeated paths) is at most $2\Delta n$. By an averaging argument, at least half of these paths have length at most

$$\frac{4\Delta n}{k'} \leq 4\Delta n \cdot \frac{20\ell}{L} \leq L$$

since $k' \geq L/(10\ell) - 2\Delta \geq L/(20\ell)$. It follows that at least a $1/(20\ell)$ proportion of the children of x are terminal. □

The following lemma shows that meta-nodes cannot have many damaged children.

Lemma 9.2.5. *Let x be a meta-node within the first ℓ levels of $\mathcal{T}$. Then at most a $1/(10\ell)$ proportion of the children of x are damaged.*

Proof. We begin with the following structural claim about alternating paths.

Claim 9.2.6. *Let $X \subseteq [\Delta + 1]$ a be set of at most 2ℓ colors and let χ and χ' be $(\Delta + 1)$-edge colorings such that the set $D := \{e \in E \mid \{\chi(e), \chi'(e)\} \setminus X \neq \emptyset, \chi(e) \neq \chi'(e)\}$ has size at most $\Delta\ell$.*

For any $(\Delta+1)$-edge coloring $\tilde{\chi}$, let $\mathcal{P}(\tilde{\chi})$ denote the set of all maximal alternating paths w.r.t. $\tilde{\chi}$ with colors in $[\Delta+1]\setminus X$. Then we have that $|\mathcal{P}(\chi)\oplus\mathcal{P}(\chi')|\leq 6\Delta^2\ell$.

Proof. Consider the following process: Start with the coloring $\chi_0 := \chi$, uncolor each of the edges in D one by one to obtain a sequence of colorings $\chi_1,\dots,\chi_{|D|}$, then recolor each of the edges in D with the color it receives under χ' to obtain a sequence of colorings $\chi_{|D|+1},\dots,\chi_{2|D|}$. Then we can observe that $\mathcal{P}(\chi')=\mathcal{P}(\chi_{2|D|})$ since any edge that receives a different color under χ' and $\chi_{2|D|}$ has colors in X under both. Furthermore, for any $0\leq i<2|D|$, we have that $|\mathcal{P}(\chi_i)\oplus\mathcal{P}(\chi_{i+1})|\leq 3\Delta$. This is because each edge can belong to at most Δ different maximal alternating paths (one for each other color) and changing the color of an edge can cause 2 maximal alternating paths to merge into 1, and vice versa. It follows that

$$|\mathcal{P}(\chi)\oplus\mathcal{P}(\chi')|\leq\sum_{i=0}^{2|D|-1}|\mathcal{P}(\chi_i)\oplus\mathcal{P}(\chi_{i+1})|\leq 6\Delta|D|\leq 6\Delta^2\ell. \qquad \square$$

Let x be a meta-node within the first ℓ levels of $\mathcal{T}$ with root-to-x path $x_0,\dots,x_i$. If x is terminal, then we are done. Otherwise, let $y_1,\dots,y_k$ denote the children of x whose types are disjoint from the types $\tau^{(x_0)},\dots,\tau^{(x_i)}$ of its ancestors (note that any child of x which is not in $\{y_1,\dots,y_k\}$ is dirty and hence not damaged). Let $e\in E$ be an edge such that $\chi^{(r)}(e)\notin\tau^{(x_0)}\cup\dots\cup\tau^{(x_i)}$ and $\chi^{(r)}(e)\neq\chi^{(y_j)}(e)$ for some y_j. Then we know that e was involved in a fan construction in one of the iterations corresponding to the meta-nodes $x_0,\dots,x_i$. Since there are at most Δ such edges per iteration, there are at most $\Delta\ell$ such edges in total. We can observe that each alternating path in the collection $P^{(y_1)},\dots,P^{(y_k)}$ appears at most twice (once in each orientation) and that each of these paths receives the same colors and is a maximal alternating path in each of the colorings $\chi^{(y_1)},\dots,\chi^{(y_k)}$. Thus, applying the preceding claim with $X=\tau^{(x_0)}\cup\dots\cup\tau^{(x_i)}$, $\chi=\chi^{(r)}$ and $\chi'=\chi^{(y_1)}$, it follows that at most $6\Delta^2\ell$ of the alternating paths in $P^{(y_1)},\dots,P^{(y_k)}$ are not maximal alternating paths with the same colors in $\chi^{(r)}$. Hence, at most $12\Delta^2\ell$ of the meta-nodes $y_1,\dots,y_k$ are damaged. Since $12\Delta^2\ell/L\leq 1/(10\ell)$, it follows that at most an $1/(10\ell)$ proportion of the children of x are damaged. $\square$

The following lemma is crucial for showing that there cannot be many long walks that do not contain any dirty or damaged meta-nodes.

Lemma 9.2.7. *Let $\tau_0,\dots,\tau_i$ be a sequence of types. Then there exist at most n meta-nodes x with root-to-x path $x_0,\dots,x_i$ such that $\tau^{(x_j)}=\tau_j$ and x_j is not dirty, damaged or terminal for all x_j.*

Proof. Let $\Gamma(\tau_0,\dots,\tau_i)$ denote the set of all such meta-nodes. We now prove the following claim.

Claim 9.2.8. *For any distinct meta-nodes $x,y\in\Gamma(\tau_0,\dots,\tau_i)$, the τ_i-alternating paths $\tilde{P}^{(x)}$ and $\tilde{P}^{(y)}$ are vertex disjoint.*

Proof. We prove this by induction. For any type τ_0 we have that $\Gamma(\tau_0)\subseteq\{x_0\}$. Thus, this claim holds trivially for $i=0$. For the inductive step, suppose that the claim holds for some

$0 \leq j-1 \leq i-1$. If the type τ_j shares a color with any of the types $\tau_0, \ldots, \tau_{j-1}$, then $\Gamma(\tau_0, \ldots, \tau_j) = \emptyset$ since the types of the meta-nodes on the root-to-x path of a non-dirty meta-node x must be disjoint. Thus, we can assume that the types $\tau_0, \ldots, \tau_j$ are disjoint. Now, let $x, y \in \Gamma(\tau_0, \ldots, \tau_j)$ and let $(u^{(x)}, v^{(x)})$ and $(u^{(y)}, v^{(y)})$ be the uncolored edges in the iterations of our algorithm corresponding to meta-nodes x and y respectively. By the induction hypothesis, the vertices $u^{(x)}$ and $u^{(y)}$ are distinct since they either lie on different positions of the same τ_{j-1}-alternating path or on different vertex disjoint paths. The τ_j-alternating paths $P^{(x)}$ and $P^{(y)}$ constructed by our algorithm have $u^{(x)}$ and $u^{(y)}$ as endpoints respectively. Since x and y are not dirty and not damaged, these alternating paths are also maximal τ_j-alternating paths with the exact same colors under $\chi^{(r)}$, so they are either distinct paths or the same path but with different orientations. In the first case, $\tilde{P}^{(x)}$ and $\tilde{P}^{(y)}$ are clearly disjoint. In the second case, we note that x and y are not terminal; hence, $P^{(x)}$ has length at least $2L+3$, so $\tilde{P}^{(x)}$ and $\tilde{P}^{(y)}$ are also vertex disjoint since they have length at most $L+1$. □

Thus, the size of $\Gamma(\tau_0, \ldots, \tau_i)$ is upper bounded by the maximum size of a collection of vertex disjoint paths, which is at most n. □

Analyzing the Random Walk

Consider the following definition of a *good* walk.

- We say that a walk $x_0, x_1, \ldots$ in the meta-tree $\mathcal{T}$ is *good* if it encounters either a terminal or contaminated meta-node within the first ℓ steps.

The following lemma bounds the probability that a good random walk has length at most $\ell + 1$.

Lemma 9.2.9. *Let $x_0, x_1, \ldots$ be a random walk in $\mathcal{T}$. Given that the random walk is good, it has length at most $\ell + 1$ with probability $1/(40\ell)$.*

Proof. Since the random walk is good, we know that x_i is terminal or contaminated for some i. If x_i is terminal, then the random walk has length $i \leq \ell$. If x_i is α-contaminated, then it follows from Lemma 9.2.3 that x_{i+1} is terminal (and hence that the random walk has length at most $i+1$) with probability at least $1/(40\ell)$. Similarly, if x_i is β-contaminated, then it follows from Lemma 9.2.4 that the random walk has length at most $i+1$ with probability at least $1/(20\ell)$. □

It follows from Lemma 9.2.9 that it suffices to lower bound the probability that a random walk in the meta-tree $\mathcal{T}$ is good. More precisely, we have the following. Let $\mathcal{W}_\mathcal{T}$ denote the collection of all walks in $\mathcal{T}$ and let $(x_i)_i \sim \mathcal{W}_\mathcal{T}$ denote a random walk. Then we have that

$$\begin{aligned}\Pr_{(x_i)_i \sim \mathcal{W}_\mathcal{T}}[|(x_i)_i| \leq \ell + 1] &\geq \Pr_{(x_i)_i \sim \mathcal{W}_\mathcal{T}}[|(x_i)_i| \leq \ell + 1 \mid (x_i)_i \text{ is good}] \cdot \Pr_{(x_i)_i \sim \mathcal{W}_\mathcal{T}}[(x_i)_i \text{ is good}] \\ &\geq \frac{1}{40\ell} \cdot \Pr_{(x_i)_i \sim \mathcal{W}_\mathcal{T}}[(x_i)_i \text{ is good}]. \end{aligned} \tag{9.1}$$

Here, the first inequality follows from conditioning on the event that the random walk $(x_i)_i$ is good, and the second inequality follows from Lemma 9.2.9.

The Meta-Subtree $\mathcal{T}'$. We define a meta-subtree $\mathcal{T}'$ of $\mathcal{T}$ as follows: Start with the meta-tree $\mathcal{T}$ and remove all of the (strict) descendants of contaminated meta-nodes. We can observe that the probability of a random walk in $\mathcal{T}$ being good is the same as the probability of a random walk in $\mathcal{T}'$ having length at most ℓ, i.e. that

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}}}[(x_i)_i \text{ is good}] = \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}'}}[|(x_i)_i| \leq \ell]. \tag{9.2}$$

To see why this is true, consider a mapping $\phi : \mathcal{W}_{\mathcal{T}} \longrightarrow \mathcal{W}_{\mathcal{T}'}$ which maps a walk $(x_i)_i$ in $\mathcal{T}$ to a walk $\phi((x_i)_i)$ which is defined as the prefix of $(x_i)_i$ which is contained in $\mathcal{T}'$. Then we can observe that $(x_i)_i$ is good if and only if $|\phi((x_i)_i)| \leq \ell$.

The Meta-Subtree $\mathcal{T}''$. We define a meta-subtree $\mathcal{T}''$ of $\mathcal{T}'$ as follows: Start with the meta-tree $\mathcal{T}'$ and remove all of the meta-nodes that are dirty or damaged (and their descendants) from $\mathcal{T}'$. It turns out that bounding the length of random walks in $\mathcal{T}''$ is much easier than bounding the length of random walks in $\mathcal{T}'$. Thus, we only want to consider random walks in $\mathcal{T}'$ which are also contained in $\mathcal{T}''$ for the first ℓ steps. In particular, we use the following bound

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}'}}[|(x_i)_i| \leq \ell] \geq \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[|(x_i)_i| \leq \ell] \cdot \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}'}}[(x_i)_{i \leq \ell} \subseteq \mathcal{T}''] . \tag{9.3}$$

Given any meta-node $x \in \mathcal{T}'$ that has children, we know that at most a $1/(5\ell)$ proportion of the children of x are dirty since x is not contaminated.[4] Furthermore, it follows from Lemma 9.2.5 that at most a $1/(10\ell)$ proportion of the children of x are damaged. It follows that a random walk in $\mathcal{T}'$ does not encounter any dirty or damaged meta-nodes within the first ℓ steps with probability at least $(1 - 3/(10\ell))^{\ell} \geq 1 - 3/10 = 7/10$. Thus, we have that

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}'}}[(x_i)_{i \leq \ell} \subseteq \mathcal{T}''] \geq \frac{7}{10}. \tag{9.4}$$

Combining these inequalities, it follows that

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}}}[|(x_i)_i| \leq \ell + 1] \geq \frac{1}{80\ell} \cdot \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[|(x_i)_i| \leq \ell]. \tag{9.5}$$

We now lower bound the probability that a random walk in $\mathcal{T}''$ terminates within at most ℓ steps.

Random Walks in $\mathcal{T}''$. Let $\mathcal{T}''_{\ell+1}$ denote the meta-nodes at depth $\ell + 1$ in $\mathcal{T}''$. Given a walk $(x_i)_i$ in $\mathcal{T}''$, $(x_i)_i$ has length greater than ℓ if and only if $(x_i)_i$ contains some meta-node from $\mathcal{T}''_{\ell+1}$, thus

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[|(x_i)_i| > \ell] = \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[\mathcal{T}''_{\ell+1} \cap (x_i)_i \neq \emptyset].$$

[4] Recall that since x is neither α nor β-contaminated, at most a $1/(10\ell)$ proportion of its children at α-dirty and at most a $1/(10\ell)$ proportion of its children at β-dirty

Lemma 9.2.10. *For any $x \in \mathcal{T}''_{\ell+1}$, we have that $\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[x \in (x_i)_i] \leq (2/L)^{\ell+1}$.*

Proof. Give some non-leaf meta-node $y \in \mathcal{T}''$, we can see that y has at least $L \cdot (1 - 3/(10\ell)) \geq L/2$ children in $\mathcal{T}''$. This is because y is not contaminated (otherwise it would be a leaf) and hence has at most $L/(5\ell)$ dirty children in $\mathcal{T}'$ and at most $L/(10\ell)$ damaged children in $\mathcal{T}'$. Thus, the probability that a random walk in $\mathcal{T}''$ contains a child y' of y given that it contains y is at most $1/(L/2)$. It follows that the probability that a random walk in $\mathcal{T}''$ contains a meta-node $x \in \mathcal{T}''_{\ell+1}$ is at most $(2/L)^{\ell+1}$. □

Lemma 9.2.11. $|\mathcal{T}''_{\ell+1}| \leq \Delta^{2\ell} nL$.

Proof. Recall the definition of Γ from the proof of Lemma 9.2.7. Let $x \in \mathcal{T}''_{\ell+1}$ and let $x_0, \ldots, x_{\ell+1}$ denote the root-to-x path in $\mathcal{T}$. Then it must be the case that $x_{\ell+1}$ is the child of some meta-node in $\Gamma(\tau^{(x_0)}, \ldots, \tau^{(x_\ell)})$ since it's parent cannot be terminal. It follows that every meta-node in $\mathcal{T}''_{\ell+1}$ is a child of some meta-node in

$$\bigcup_{\tau \in \binom{[\Delta+1]}{2}^{\ell}} \Gamma(\tau).$$

Since any meta-node has at most L children and $|\Gamma(\tau)| \leq n$ by Lemma 9.2.7, it follows that

$$|\mathcal{T}''_{\ell+1}| \leq L \cdot \sum_{\tau \in \binom{[\Delta+1]}{2}^{\ell}} |\Gamma(\tau)| \leq L \cdot (\Delta^2)^{\ell} \cdot n \leq \Delta^{2\ell} nL.$$ □

It follows from Lemmas 9.2.10 and 9.2.11 that

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[\mathcal{T}''_{\ell+1} \cap (x_i)_i \neq \emptyset] \leq \sum_{x \in \mathcal{T}''_{\ell+1}} \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[x \in (x_i)_i] \leq |\mathcal{T}''_{\ell+1}| \cdot \left(\frac{2}{L}\right)^{\ell+1} \leq nL \cdot \left(\frac{2\Delta^2}{L}\right)^{\ell+1} \leq \frac{1}{2}.$$

Hence, we have that

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[|(x_i)_i| \leq \ell] = 1 - \Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}''}}[|(x_i)_i| > \ell] \geq \frac{1}{2}. \tag{9.6}$$

Combining Equations (9.5) and (9.6), it follows that

$$\Pr_{(x_i)_i \sim \mathcal{W}_{\mathcal{T}}}[|(x_i)_i| \leq \ell + 1] \geq \frac{1}{160\ell}. \tag{9.7}$$

In other words, a random walk in the meta-tree $\mathcal{T}$ terminates within $\ell + 1$ steps with probability at least $1/(80\ell)$.

9.2.2 Proof of Theorem 9.2.1

Suppose that we run Algorithm 19 for $\kappa = (\ell+1) \cdot 1600 \log^2 n$ iterations. It follows from Lemma 9.2.2 that, given an arbitrary input, the probability of the algorithm successfully extending the coloring

within $\ell + 1$ iterations is at least $1/(160 \log n)$. Thus, we can split these κ updates into batches of $\ell + 1$ iterations, where the probability that the algorithm successfully extends χ within some batch is at least $1/(160 \log n)$. Thus, the probability that the algorithm fails to extend the coloring within κ iterations it at most

$$\left(1 - \frac{1}{160 \log n}\right)^{1600 \log^2 n} \leq \exp\left(-\frac{1600 \log^2 n}{160 \log n}\right) = \frac{1}{n^{10}}.$$

In the low probability event that the algorithm does not successfully extend the coloring within κ iterations, then we can extend it in $O(n)$ time using Vizing's original algorithm. Thus, our algorithm runs in time $\kappa \cdot \tilde{O}(L) \leq \tilde{O}(\Delta^2 + \sqrt{\Delta n})$ both with high probability and in expectation.

Chapter 10

Dynamic Edge Coloring II: Nibbling at Long Cycles

In this chapter, we give our algorithms for $(1+\epsilon)\Delta$-coloring based on the Nibble method. As our main result in this chapter, we prove Theorem 1.1.8, which we restate below.

Theorem 1.1.8. *There is an algorithm that, given a dynamic graph G with maximum degree at most $\Delta \geq (\log n/\epsilon)^{\text{poly}(1/\epsilon)}$, maintains a $(1+\epsilon)\Delta$-coloring of G with $\text{poly}(1/\epsilon)$ expected worst-case update time, against an oblivious adversary.*

As a corollary of this result, we obtain our static $(1+\epsilon)\Delta$-coloring algorithm, proving Corollary 1.1.9.

Corollary 1.1.9. *There is an algorithm that, given a graph G with maximum degree at most $\Delta \geq (\log n/\epsilon)^{\text{poly}(1/\epsilon)}$, computes a $(1+\epsilon)\Delta$-coloring of G in $O(m\,\text{poly}(1/\epsilon))$ time with high probability.*

Notation. This chapter uses completely different techniques from the algorithms in Chapters 7 to 9, and does not use the notion of Vizing chains, Vizing fans, or alternating paths. Consequently, this chapter uses different notation. This chapter is fully self contained. We introduce the main notation used throughout this chapter while describing our static algorithm in Section 10.1.1. In Section 10.2, we describe the notation for our dynamic algorithm.

10.1 Our Static Algorithm

Throughout this chapter, we fix some constant ϵ such that $0 < \epsilon \leq 1/10$, and define parameters $T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ and $K := \lceil (8/\epsilon^2)\log(1/\epsilon) \rceil$. In this section, we describe our static algorithm in full detail and provide a complete analysis. The main result in this section is Theorem 10.1.2, which is restated below.

Theorem 10.1.1. *Let $\epsilon \in (0, 1/10)$ be a constant. Then, given a graph G and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ, the algorithm StaticColor produces a $(1+61\epsilon)\Delta$-edge coloring of G with probability at least $1 - 8/n^6$.*

10.1.1 Algorithm Description

The Algorithm StaticColor**.** Our main algorithm, StaticColor, takes as input a graph $G = (V, E)$ with n nodes, and a parameter $\Delta \in \mathbb{N}$. The algorithm uses two subroutines: Partition and Nibble. We first use the algorithm Partition to split the input graph G into $\eta = \lceil \Delta^{1-1/(30T)} \rceil$ many graphs $\mathcal{G}_1, ..., \mathcal{G}_\eta$ by assigning the edges of G to one of the $\mathcal{G}_j$ independently and uniformly at random. We then proceed to call the algorithm Nibble on each of the $\mathcal{G}_j$, where we use $(1+\epsilon)^2\Delta^{1/(30T)}$ many colors to color most of the edges in $\mathcal{G}_j$. Finally, we take all of the edges that failed to be colored by our calls to Nibble and greedily color them. Algorithm 20 gives the pseudocode for StaticColor.

Algorithm 20: StaticColor(G, ϵ)

$\mathcal{G}_1, \ldots, \mathcal{G}_\eta \leftarrow$ Partition$(G, \Delta, \Delta^{1/(30T)})$
for $j = 1, ..., \eta$ **do**
  $\mathcal{F}_j \leftarrow$ Nibble$(\mathcal{G}_j, (1+\epsilon)\Delta^{1/(30T)}, \epsilon)$
Greedily color the edges in $H \leftarrow \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_\eta$ using $3\Delta(H)$ many colors

For analytic purposes, and also to make the algorithm easier to present and dynamize, it will be useful for us to fix the randomness used by our algorithm while making calls to Partition and Nibble in advance. Hence, we will describe how all the relevant randomness is generated in advance while describing each part of our algorithm. The assumption can be removed later.

The Algorithm Partition**.** This algorithm takes as input a graph G and some parameters Δ and Δ' such that $\Delta' \leq \Delta$, and outputs a partition $\mathcal{G}_1, ..., \mathcal{G}_\eta$ of G, where $\eta = \lceil \Delta/\Delta' \rceil$, obtained by assigning the edges of G to one of the $\mathcal{G}_j$ independently and uniformly at random. The node sets $V(\mathcal{G}_j)$ in all of the $\mathcal{G}_j$ are the same as the node set V of the graph G. In order to fix the randomness in advance, for each *potential edge* of the graph $e \in \binom{V}{2}$, we sample an index $j_e \in [\eta]$ independently and uniformly at random, and place e into the graph $\mathcal{G}_{j_e}$. Notice that after fixing the random variables $\{j_e\}_e$ the partition of the graph depends only on what edges are present in G. Algorithm 21 gives the pseudocode for Partition.

Algorithm 21: Partition(G, Δ, Δ')

$\eta \leftarrow \lceil \Delta/\Delta' \rceil$
$\mathcal{E}_j \leftarrow \emptyset$ for all $j \in [\eta]$
for $e \in E$ **do**
  Place e into one of $\mathcal{E}_1, \ldots, \mathcal{E}_\eta$ independently and u.a.r.
$\mathcal{G}_j \leftarrow (V, \mathcal{E}_j)$ for all $j \in [\eta]$
return $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$

The Algorithm Nibble**.** This algorithm takes as input a graph G, a parameter Δ, and uses a palette $\mathcal{C}$ of $\lceil (1+\epsilon)\Delta \rceil$ colors. The Nibble algorithm runs for $T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ *rounds*.

At the start of round $i \in [T]$, we have a subset of edges $E_i \subseteq E$ such that the algorithm has already assigned tentative colors to the remaining edges $E \setminus E_i$. We denote this *tentative* partial coloring by $\tilde{\chi} : E \setminus E_i \to \mathcal{C} \cup \{\perp\}$, which need not necessarily be proper. For each node $u \in V$, we

refer to the set of colors $P_i(u) := \mathcal{C} \setminus \tilde{\chi}(N(u) \setminus E_i)$ as the *palette* of u at the start of round i, where $N(u) \subseteq E$ denotes the set of edges incident on u in G. In words, the palette $P_i(u)$ consists of the set of colors that were *not* tentatively assigned to any incident edge of u in previous rounds. We also define $P_i(u,v) := P_i(u) \cap P_i(v)$ to be the *palette* of any edge $(u,v) \in E_i$ at the start of round i.

We start by initializing $E_1 \leftarrow E$, and $P_1(u) \leftarrow \mathcal{C}$ for all $u \in V$. Subsequently, for $i = 1, \ldots, T$, we implement round i as follows. Each edge $e \in E_i$ *selects* itself independently with probability ϵ. Let $S_i \subseteq E_i$ be the set of selected edges. Next, in parallel, each edge $e \in S_i$ samples $K := \lceil (8/\epsilon^2) \log(1/\epsilon) \rceil$ colors $c_1, \ldots, c_K$ independently and uniformly at random from $\mathcal{C}$ and sets its tentative color $\tilde{\chi}(e)$ to some c_ℓ which is contained in $P_i(e)$ if such a color exists. Otherwise, we set $\tilde{\chi}(e) \leftarrow \perp$. At this point, we define the collection $F_i \subseteq S_i$ of *failed* edges in round i. We say that an edge $e = (u,v) \in S_i$ *fails* in round i iff either (i) $\tilde{\chi}(e) = \perp$, or (ii) there is a neighboring edge $f \in (N(u) \cup N(v)) \cap S_i$ which was also selected in round i and received the same tentative color as the edge e (i.e., $\tilde{\chi}(e) = \tilde{\chi}(f)$). Let $F_i \subseteq S_i$ denote this collection of failed edges (in round i). We now set $E_{i+1} \leftarrow E_i \setminus S_i$ and proceed to the next round $i+1$.

In order to fix the randomness in advance, for each potential edge of the graph e, we sample a round for the edge e independently from $\textsc{CappedGeo}(\epsilon, T+1)$, which we denote by i_e.[1] We also assume that for each potential edge e we have some colors $c_e(1), ..., c_e(K)$ where each $c_e(\ell)$ is sampled uniformly at random and independently from $\mathcal{C}$. We define ℓ_e to be $\min\{\ell \mid c_e(\ell) \in P_{i_e}(e)\}$ (taking the convention that $\min \emptyset = 0$) and note that $\tilde{\chi}(e) = c_e(\ell_e)$ (as long as $c_e \cap P_{i_e}(e) \neq \emptyset$). Algorithm 22 gives the pseudocode for Nibble.

To ease notations, at the end of the last round T we define $F_{T+1} \leftarrow E_{T+1}$, and $\tilde{\chi}(e) \leftarrow \perp$ for all $e \in F_{T+1}$. We let $F := \bigcup_{i=1}^{T+1} F_i$ denote the set of failed edges across all the rounds. We denote $N(u) \cap S_i$ by $N_i(u)$. It is easy to check that the tentative coloring $\tilde{\chi}$, when restricted to the edge-set $E \setminus F$, is already proper.

[1] Thus, we have $\Pr[i_e = i] = (1-\epsilon)^{i-1}\epsilon$ for all $i \in [T]$ and $\Pr[i_e = T+1] = (1-\epsilon)^T$.

Algorithm 22: NIBBLE(G, Δ, ϵ)

$\chi(e) \leftarrow \perp$ and $\tilde{\chi}(e) \leftarrow \perp$ for all $e \in E(G)$
$E_1 \leftarrow E(G)$
for $i = 1, ..., T$ **do**
  for $e \in S_i$ **do**
    $\ell_e \leftarrow \min\{\ell \mid c_e(\ell) \in P_i(e)\}$
    if $1 \leq \ell_e \leq K$ **then**
      $\tilde{\chi}(e) \leftarrow c_e(\ell_e)$
  $F_i \leftarrow \{e \in S_i \mid \exists f \in N(e) \cap S_i \text{ such that } \tilde{\chi}(f) = \tilde{\chi}(e)\} \cup \{e \in S_i \mid \tilde{\chi}(e) = \perp\}$
  $\chi(e) \leftarrow \tilde{\chi}(e)$ for all $e \in S_i \setminus F_i$
  $E_{i+1} \leftarrow E_i \setminus S_i$
$F_{T+1} \leftarrow E_{T+1}$
$F \leftarrow \bigcup_{i=1}^{T+1} F_i$
return F

We now prove the following result which describes the behavior of StaticColor.

Theorem 10.1.2. *Let $\epsilon \in (0, 1/10)$ be a constant. Then, given a graph G and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ, the algorithm StaticColor produces a $(1+61\epsilon)\Delta$-edge coloring of G with probability at least $1 - 8/n^6$.*

10.1.2 Analysis on Locally Treelike Graphs

For the rest of Section 10.1.2, fix some graph $G = (V, E)$ of maximum degree at most Δ such that $\Delta \geq (100 \log n)/\epsilon^4$. Let $\mathcal{N}_G(u, j) := G[\{v \in V : \text{dist}_G(u, v) \leq j\}]$ denote the j-hop neighborhood of any node $u \in V$.[2] We refer to $\mathcal{N}_G(u, j)$ as the j-neighborhood of u. Let $U \subseteq V$ be the set of nodes $u \in V$ such that the $(T+1)$-neighborhood of u is a tree. We refer to the nodes in U as *good* and the nodes in $V \setminus U$ as *bad*. Now suppose we run Algorithm 22 on input (G, Δ, ϵ). We now proceed to analyze the behavior of the algorithm on this input.

Locality of the Nibble Method

Lemma 10.1.3. *Let $u \in V$, $i \in [T]$. Then $P_i(u)$ depends only on $\mathcal{N}(u, i-1)$.*

Proof. We first recall the way our algorithm generates random bits in advance. We assign each *potential edge* $e \in \binom{V}{2}$ a round i_e sampled from CappedGeo$(\epsilon, T+1)$ independently. Furthermore, we assign each potential edge a sequence of colors $c_e(1), \ldots c_e(K)$ generated by sampling colors independently and u.a.r from $\mathcal{C}$. We now argue inductively that the set $P_i(u)$ depends only on $\mathcal{N}(u, i-1)$, i.e. that the structure of the graph outside of the $(i-1)$-neighborhood of u has no effect on the palette of u during the first $i-1$ rounds of the nibble method.

[2] We use the symbole $\text{dist}_G(u, v)$ to denote the distance between u and v in G. Furthermore, for any subset of nodes $V' \subseteq V$, the symbol $G[V']$ denotes the subgraph of G induced by V' and $\mathcal{N}_G(V', j)$ denotes $\bigcup_{u \in V'} \mathcal{N}_G(u, j)$.

Clearly, $P_1(u) = \mathcal{C}$ for all nodes u and hence does not depend on anything. Now, for the induction step, suppose that $P_i(u)$ depends only on $\mathcal{N}(u, i-1)$ for all nodes u. Given some node u, note that $P_{i+1}(u)$ depends on (i) the rounds of the edges incident on u, (ii) the sequences of colors assigned to the edges incident on u, and (iii) the palettes $P_i(v)$ of the nodes adjacent to u. Since (i) and (ii) are fixed in advance, $P_{i+1}(u)$ is completely determined by which nodes are its neighbors and their palettes at the end of iteration i. Hence, by the induction hypothesis, it follows that $P_{i+1}(u)$ depends on $\bigcup_{v \in N(u)} \mathcal{N}(v, i-1)$ which is exactly the i-neighborhood of u, $\mathcal{N}(u, i)$. □

It immediately follows from Lemma 10.1.3 that, for an edge e, the palette $P_i(e)$ depends only on $\mathcal{N}(e, i-1)$. Since the tentative color assigned to e depends only on the round i_e and the palette $P_{i_e}(e)$, it also follows that $\tilde{\chi}(e)$ depends only on $\mathcal{N}(e, i_e - 1) \subseteq \mathcal{N}(e, T-1)$. Hence, given some node u, the tentative colors assigned to edges in $N(u)$ depend only on $\mathcal{N}(N(u), T-1) \subseteq \mathcal{N}(u, T)$. Since the edges in $N(u)$ that fail depend on the tentative colors of the edges in $N^2(U) := N(N(U))$, while analyzing the palettes, tentative colors, and failure status of the edges in $N(u)$ we can assume that the input graph is $\mathcal{N}(N^2(u), T-1) = \mathcal{N}(u, T+1)$. In the case where u is good, this will allow us to significantly simplify the analysis.

The Symmetry of Algorithm 6

We now prove some lemmas that will be crucial for analyzing the types of colorings generated by our algorithm. For the rest of Section 10.1.2 we fix the random bits that determine the rounds of all edges in G. The following lemma formalizes the main "symmetry" property of Nibble which says that it treats all colors equally.

Lemma 10.1.4. *For all $u \in V$, $i \in [T]$, $C \subseteq \mathcal{C}$, and permutations $\pi : \mathcal{C} \longrightarrow \mathcal{C}$, we have that*

$$\Pr\left[P_i(u) = C\right] = \Pr\left[\pi\left(P_i(u)\right) = C\right].$$

Proof. We prove this lemma via a coupling argument. Let $\mathcal{A}$ denote the algorithm Nibble. Given some permutation π of $\mathcal{C}$, we define a new algorithm $\mathcal{A}^\pi$, which behaves in the exact same way as $\mathcal{A}$, except that given the color sequence c_e for the potential edge e, it uses the color sequences $\pi \circ c_e$ instead. We denote the palette $P_i(u)$ produced by algorithm $\mathcal{A}'$ by $P_i^{(\mathcal{A}')}(u)$. We now prove by induction that for all $u \in V$, $i \in [T]$, and permutations π of $\mathcal{C}$, we have that

$$P_i^{(\mathcal{A}^\pi)}(u) = \pi\left(P_i^{(\mathcal{A})}(u)\right). \tag{10.1}$$

Fix some permutation π of $\mathcal{C}$. It's easy to see that (10.1) holds for all $u \in V$ when $i = 1$ since the palettes all equal $\mathcal{C}$ and π is a permutation. Now suppose that (10.1) holds for some $i \in [T-1]$ and all $u \in V$. Then, for all $e = (u, v) \in S_i$, we have that

$$P_i^{(\mathcal{A}^\pi)}(e) = P_i^{(\mathcal{A}^\pi)}(u) \cap P_i^{(\mathcal{A}^\pi)}(v) = \pi\left(P_i^{(\mathcal{A})}(u)\right) \cap \pi\left(P_i^{(\mathcal{A})}(v)\right)$$

$$= \pi\left(P_i^{(\mathcal{A})}(u) \cap P_i^{(\mathcal{A})}(v)\right) = \pi\left(P_i^{(\mathcal{A})}(e)\right),$$

thus, it follows that

$$c_e(\ell) \in P_i^{(\mathcal{A})}(e) \text{ iff } \pi(c_e(\ell)) \in \pi\left(P_i^{(\mathcal{A})}(e)\right) \text{ iff } \pi(c_e(\ell)) \in P_i^{(\mathcal{A}^\pi)}(e).$$

We get that $\ell_e^{(\mathcal{A})} = \ell_e^{(\mathcal{A}^\pi)}$ for all $e \in S_i$, which implies that

$$\tilde{\chi}^{(\mathcal{A}^\pi)}(e) = \pi\left(c_e(\ell_e^{(\mathcal{A}^\pi)})\right) = \pi\left(c_e(\ell_e^{(\mathcal{A})})\right) = \pi\left(\tilde{\chi}^{(\mathcal{A})}(e)\right)$$

for all $e \in S_i$.[3] Finally, it follows that

$$\pi\left(P_{i+1}^{(\mathcal{A})}(u)\right) = \pi\left(P_i^{(\mathcal{A})}(u) \setminus \tilde{\chi}^{(\mathcal{A})}(N_i(u))\right) = \pi\left(P_i^{(\mathcal{A})}(u)\right) \setminus \pi\left(\tilde{\chi}^{(\mathcal{A})}(N_i(u))\right)$$

$$= P_i^{(\mathcal{A}^\pi)}(u) \setminus \tilde{\chi}^{(\mathcal{A}^\pi)}(N_i(u)) = P_{i+1}^{(\mathcal{A}^\pi)}(u).$$

This concludes the induction. By now noticing that algorithms $\mathcal{A}$ and $\mathcal{A}^\pi$ are actually the same (since applying a permutation to an independent uniform sample returns an independent uniform sample) we get that, for any $C \subseteq \mathcal{C}$,

$$\Pr\left[P_i^{(\mathcal{A})}(u) = C\right] = \Pr\left[P_i^{(\mathcal{A}^\pi)}(u) = C\right] = \Pr\left[\pi\left(P_i^{(\mathcal{A})}(u)\right) = C\right]$$

and the lemma follows. □

Fix any $i \in [T]$ and let X_c^w be the indicator for the event that $c \in P_i(w)$ for a color c and node w.

Lemma 10.1.5. *Let $u \in V$. Given that $\sum_c X_c^u = \gamma$ for some $\gamma \in \mathbb{N}$, we have that $\{X_c^u\}_c$ is a permutation distribution (see Definition 10.5.7) where exactly γ of the X_c^u are 1 and the rest are 0.*

Proof. Since the random variables X_c^u are all indicators and we know that $\sum_c X_c^u = \gamma$, it follows that exactly γ of them are 1 and the rest are 0. It then follows from Lemma 10.1.4 that, for any $C, C' \subseteq \mathcal{C}$ of size γ, $\Pr[P_i(u) = C] = \Pr[P_i(u) = C']$, and hence this collection of random variables forms a permutation distribution. □

Lemma 10.1.6. *Let $\mathfrak{X}_1, \ldots, \mathfrak{X}_\ell$ be families of random variables that depend on disjoint collections of random bits. Then these families of random variables are mutually independent.*

Lemma 10.1.7. *Let $u_1, \ldots, u_\ell \in V$ be nodes that are mutually disconnected in the graph $(V, S_{<i})$. Then we have that the families of random variables $\{X_c^{u_1}\}_c, \ldots, \{X_c^{u_\ell}\}_c$ depend on disjoint collections of random bits.*

Lemma 10.1.8. *Let $e = (u, v) \in S_i$, such that $e \in N^2(U)$ and u and v are not connected in the graph $(V, S_{<i})$. Suppose we fix the random bits used in the first $i - 1$ rounds that determine the*

[3] Here $\ell_e^{(\mathcal{A}')}$ and $\tilde{\chi}^{(\mathcal{A}')}(e)$ denote the color index ℓ_e and tentative color $\tilde{\chi}(e)$ produced by algorithm $\mathcal{A}'$.

palette $P_i(u)$ (but not those that determine $P_i(v)$) and let c be a color that does not depend on the random bits that determine $P_i(u)$. Then we have that $\Pr[\tilde{\chi}(e) = c] \leq 1/|P_i(u)|$.

Proof. First, note that by Lemma 10.1.3 we can assume that the input graph is $\mathcal{N}(e, T-1)$. Since $e \in N^2(U)$, $\mathcal{N}(e, T-1)$ is a subgraph of $\mathcal{N}(w, T+1)$ for some $w \in U$, and hence $\mathcal{N}(e, T-1)$ is a tree. It follows that u and v are not connected in the graph $(V, S_{<i})$. Note that since $P_i(u)$ and $P_i(v)$ are functions of $\{X_c^u\}_c$ and $\{X_c^v\}_c$ respectively, which depend on disjoint sets of random bits by Lemma 10.1.7, it is possible to fix the random bits this way. In the event that e does not fail, $\tilde{\chi}(e)$ is a uniform sample from $P_i(u) \cap P_i(v)$ by the construction of our algorithm. By Lemma 10.1.6, the families of random variables $\{X_c^u\}_c$ and $\{X_c^v\}_c$ are independent, so by Lemma 10.1.5 we get that $P_i(v)$ is a uniform random subset of $\mathcal{C}$. Hence, as long as $P_i(u) \cap P_i(v) \neq \emptyset$, sampling a color uniformly at random from $P_i(u) \cap P_i(v)$ is the same as sampling a color uniformly at random from $P_i(u)$. It follows that

$$\Pr[\tilde{\chi}(e) = c] \leq \Pr[\tilde{\chi}(e) = c \,|\, c \in P_i(u)] \leq \frac{1}{|P_i(u)|}. \qquad \square$$

Concentration of Basic Quantities

We now establish the concentration of some basic quantities. We first start by analyzing how many edges incident on a node are sampled during a round. For $u \in V$ and $i \in [T]$, let $N_i(u)$ denote the set of edges $N(u) \cap S_i$, and let $N_{\geq i}(u)$ denote the set of edges $\cup_{i' \geq i} N_{i'}(u)$.

Lemma 10.1.9. *For all $u \in V$, $i \in [T]$, we have that*

$$|N_i(u)| < (\epsilon + \epsilon^2)(1-\epsilon)^{i-1}\Delta$$

with probability at least $1 - 1/n^{14}$.

Proof. Let $u \in V$ and $i \in [T]$. Since the round of each edge is sampled from the capped geometric distribution, it follows that

$$\mathbb{E}[|N_i(u)|] = \sum_{e \in N(u)} \Pr[e \in S_i] = \epsilon(1-\epsilon)^{i-1}|N(u)| \leq \epsilon(1-\epsilon)^{i-1}\Delta.$$

Since the rounds of edges are sampled independently, we can apply a Chernoff bound to get concentration. It follows that

$$\Pr\left[|N_i(u)| \geq (1+\epsilon) \cdot \epsilon(1-\epsilon)^{i-1}\Delta\right] \leq \exp(-\epsilon^2 \cdot \epsilon(1-\epsilon)^{i-1}\Delta/3).$$

Now note that

$$(1-\epsilon)^{i-1} \geq e^{-\epsilon T}(1-\epsilon^2 T) \geq \epsilon(1-\epsilon\log(1/\epsilon)) \geq \epsilon/2,$$

and hence we have that

$$\exp(-\epsilon^2 \cdot \epsilon(1-\epsilon)^{i-1}\Delta/3) \leq \exp(-\epsilon^4\Delta/6) \leq \exp(-16\log n) = 1/n^{16}.$$

The result follows by union bounding over all $u \in V$, $i \in [T]$, and noting that $T \leq 1/\epsilon^4 \leq n$. □

We now define an event $\mathcal{Z}$ which occurs if and only if $|N_i(u)| < (\epsilon+\epsilon^2)(1-\epsilon)^{i-1}\Delta$ for all $u \in V$, $i \in [T]$. By Lemma 10.1.9, this event occurs with probability at least $1 - 1/n^{14}$. For the rest of Section 10.1.2, we fix all of the random bits used to determine the rounds of the potential edges and assume that event $\mathcal{Z}$ occurs. We implicitly condition all probabilities on event $\mathcal{Z}$ unless explicitly stated otherwise. Hence, when taking expectations and probabilities, we are doing so over the randomness of the color sequences assigned to edges.

Lemma 10.1.10. *For all $u \in V$, $i \in [T]$, we have that*

$$|P_i(u)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta.$$

Proof. Given any $u \in V$, $i \in [T]$, we have that

$$|N_{<i}(u)| = \sum_{j=1}^{i-1} |N_j(u)| < (\epsilon+\epsilon^2)\Delta \sum_{j=1}^{i-1}(1-\epsilon)^{j-1} = (1+\epsilon)\Delta \cdot (1-(1-\epsilon)^{i-1}).$$

It follows that $|P_i(u)| \geq (1+\epsilon)\Delta - |N_{<i}(u)| \geq (1+\epsilon)(1-\epsilon)^{i-1}\Delta$. □

Lemma 10.1.11. *For all $e \in E$, $i \in [T]$ such that $e \in E_i$ and $e \in N^2(U)$, we have that*

$$|P_i(e)| > (1-\epsilon^2)(1-\epsilon)^{2(i-1)}\Delta$$

with probability at least $1 - 1/n^9$.

Proof. Given any such edge $e = (u, v)$, we have that

$$|P_i(e)| = \sum_{c \in [(1+\epsilon)\Delta]} X_c^u \cdot X_c^v$$

where X_c^w is the indicator for the event that $c \in P_i(w)$ for a color c and a node w. Since one of the endpoints of e is distance at most 2 from a good node, we get that $\mathcal{N}(e, T-1)$ is a tree. Since we only want to analyse the palette $P_i(e)$, we can assume that the input graph is $\mathcal{N}(e, T-1)$, which is a tree. Hence, the connected components containing u and v in the graph $(V, S_{<i})$ are not connected, so by Lemmas 10.1.6 and 10.1.7 it follows that $\{X_c^u\}_c$ and $\{X_c^v\}_c$ are independent families of random variables. Note that, by Lemma 10.1.5, given that $|P_i(u)| = \gamma$ for some $\gamma \in \mathbb{N}$, $\{X_c^u\}_c$ is a permutation distribution where exactly γ of the X_c^u are 1 and the rest are 0. Since we know that $|P_i(v)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta$, we can define a collection of random variables $\{Y_c^u\}_c$ by taking a uniform random subset Π^u of size $(1+\epsilon)(1-\epsilon)^{i-1}\Delta$ of the set $\{c \in [(1+\epsilon)\Delta] \mid X_c^u = 1\}$ and letting Y_c^u indicate

whether $c \in \Pi^u$. It follows that $\{Y_c^u\}_c$ is a permutation distribution where exactly $(1+\epsilon)(1-\epsilon)^{i-1}\Delta$ of the Y_u^c are 1, and hence by Proposition 10.5.8 is an NA collection of indicator random variables such that $Y_c^u \leq X_c^u$ for each c. We define $\{Y_c^v\}_c$ in the exact same way. Since $\{X_c^u\}_c$ and $\{X_c^v\}_c$ are independent families, it follows that $\{Y_c^u\}_c$ and $\{Y_c^v\}_c$ are independent families, and we can apply closure under products (Proposition 10.5.9) to get that $\{Y_c^u, Y_c^v\}_c$ is also a family of NA random variables. By then applying disjoint monotone aggregation (Proposition 10.5.9), it follows that $\{Y_c^u \cdot Y_c^v\}_c$ are NA. Hence, we can apply a Chernoff bound. Letting $Y_c = Y_c^u \cdot Y_c^v$, we first note that

$$\mathbb{E}\left[\sum_c Y_c\right] = \sum_c \mathbb{E}\left[Y_c^u\right] \cdot \mathbb{E}\left[Y_c^v\right] = (1+\epsilon)\Delta \cdot \frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)\Delta} \cdot \frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)\Delta}$$

$$= (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta.$$

It follows that

$$\Pr\left[\sum_c Y_c \leq (1-\epsilon)\cdot(1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta\right] \leq \exp(-\epsilon^2 \cdot (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta/2).$$

As we saw in the proof of Lemma 10.1.9, $(1-\epsilon)^{i-1} \geq \epsilon/2$, so it follows that

$$\exp(-\epsilon^2 \cdot (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta/2) \leq \exp(-\epsilon^4\Delta/8) \leq \exp(-12\log n) \leq 1/n^{12}.$$

The result follows by union bounding over all $e \in E$, $i \in [T]$, and noting that $|P_i(e)| \geq \sum_c Y_c$. □

Analyzing the Failed Edges

Given some good node $u \in U$, we now bound the number of edges incident on u that fail to be colored, either because the algorithm failed to find a color in its palette or because of conflicts with neighboring edges. For $u \in V$, $i \in [T+1]$, we denote by $F_i(u)$ the set $N_i(u) \cap F_i$ of edges incident on u that fail during iteration i. We begin by bounding the number of edges in $F_{T+1}(u)$, and then proceed to bound $F_i(u)$ for $i \in [T]$. The following lemma is *not* conditioned on event $\mathcal{Z}$.

Lemma 10.1.12. *Let $u \in V$. Then we have that $|F_{T+1}(u)| \leq \epsilon(1+\epsilon)\Delta$ with probability at least $1 - 1/n^{33}$.*

Proof. Let $u \in V$. Given some $e \in N(u)$, we can see that the probability that e is never selected to be colored during a round is $(1-\epsilon)^T$. Since all of these edges are sampled to be colored independently, it follows that

$$\mathbb{E}[|F_{T+1}(u)|] = (1-\epsilon)^T|N(u)| \leq e^{-\epsilon T}\Delta = \epsilon\Delta.$$

By applying Chernoff bounds, it follows that

$$\Pr\left[|F_{T+1}(u)| \geq (1+\epsilon)\cdot\epsilon\Delta\right] \leq \exp(-\epsilon\Delta\cdot\epsilon^2/3) \leq \exp(-33\log n) = 1/n^{33}.$$ □

For the rest of Section 10.1.2, we again fix all of the random bits used to determine the rounds of the potential edges and assume that event $\mathcal{Z}$ occurs, implicitly conditioning all probabilities on $\mathcal{Z}$.

For some $i \in [T]$, we now categorize the failed edges in $F_i(u)$ into 3 types and bound each of these individually. We say that edges e and f *conflict* if they share an endpoint and are assigned the same tentative color. Note that conflicting edges must receive their tentative colors on the same round. Given some edge $e \in F_i(u)$, we place e into $F_i'(u)$ iff there exists some edge $f \in F_i(u)$ such that e and f conflict and we place e into $F_i''(u)$ iff there exists some edge $f \in F_i \setminus F_i(u)$ such that e and f conflict. $F_i'(u) \cup F_i''(u)$ capture all of the edges that fail due to conflicts, i.e. $F_i'(u) \cup F_i''(u) = \{e \in N_i(u) \mid \exists f \in N_i(e) \text{ such that } \tilde{\chi}(e) = \tilde{\chi}(f)\}$. Finally, we let $F_i'''(u)$ be the edges $e \in F_i(u)$ that fail because the algorithm does not find a color in its palette, i.e. $c_e \cap P_i(e) = \emptyset$. Clearly $F_i(u) = F_i'(u) \cup F_i''(u) \cup F_i'''(u)$. Note that these sets are not necessarily disjoint. We now proceed to bound $|F_i(u)|$ by individually bounding $|F_i'(u)|$, $|F_i''(u)|$ and $|F_i'''(u)|$. We split up the bound in this way because the techniques required to establish concentration on the sizes of these sets are slightly different.

Lemma 10.1.13. *Let $u \in U$, $i \in [T]$. Then we have that $|F_i'(u)| \leq 2(\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at least $1 - 1/n^{25}$.*

Proof. Let $u \in U$, $i \in [T]$, and arbitrarily fix all of the random bits used in the first $i-1$ rounds that determine the palette $P_i(u)$. Let $u_1, \ldots, u_\ell$ be the nodes that are connected to u by an edge in $N_i(u)$. Recall that we can assume the input graph is $\mathcal{N}(u, T+1)$, which is a tree. Since the graph is a tree, the nodes $u, u_1, \ldots, u_\ell$ are all disconnected from each other in $(V, S_{<i})$, and hence $\{X_{u_1}^c\}_c, \ldots, \{X_{u_\ell}^c\}_c$ are mutually independent families of random variables by Lemmas 10.1.6 and 10.1.7.

Let $e \in N_i(u)$. Given some $f \in N_i(u) \setminus \{e\}$, the probability that e and f conflict is the probability that e and f are assigned the same color. Letting $c = \tilde{\chi}(e)$, we have by Lemma 10.1.8 that $\Pr[\tilde{\chi}(f) = c] \leq 1/|P_i(u)|$. Let $f, f' \in N_i(u) \setminus \{e\}$ be distinct edges such that $f = (u, v)$ and $f = (u, v')$. Then the event $\tilde{\chi}(f) = c$ depends on the random bits that determine $P_i(v)$ and the random bits used to sample $\tilde{\chi}(f)$ (recall that the bits that determine $P_i(u)$ are fixed). Hence, by Lemma 10.1.6, the events $\tilde{\chi}(f) = c$ and $\tilde{\chi}(f') = c$ are independent since they depend on distinct random bits. It follows that

$$\Pr[e \notin F_i'(u)] = \prod_{f \in N_i(u) \setminus \{e\}} \Pr\left[\tilde{\chi}(f) \neq c\right] \geq \prod_{f \in N_i(u) \setminus \{e\}} \left(1 - \frac{1}{|P_i(u)|}\right)$$

$$\geq \left(1 - \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}\right)^{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta} \geq 1 - \frac{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta} = 1 - \epsilon,$$

where the last inequality follows from Bernoulli's inequality. By linearity of expectation, we get that

$$\mathbb{E}[|F_i'(u)|] = \sum_{e \in N_i(u)} \Pr[e \in F_i'(u)] \leq \epsilon |N_i(u)| \leq (\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta.$$

We now establish concentration. Let $N_i(u) = \{e_1, ..., e_\ell\}$ and $\phi'(\tilde{\chi}(e_1), ..., \tilde{\chi}(e_\ell))$ be the function that counts the number of edges in $F_i'(u)$, i.e. $\phi' = |F_i'(u)|$. Since the function ϕ' is Lipschitz with all constants 2 (Definition 10.5.3) and $\tilde{\chi}(e_1), ..., \tilde{\chi}(e_\ell) \in P_i(u)$ are mutually independent as they depend on distinct random bits, we can apply the method of bounded differences (Proposition 10.5.4) to get that

$$\Pr\left[\phi' \geq \mathbb{E}[\phi'] + t\right] \leq \exp\left(-\frac{t^2}{2|N_i(u)|}\right)$$

for all $t > 0$. By setting $t = \epsilon^2(1+\epsilon)(1-\epsilon)^{i-1}\Delta$ we get that $|F_i'(u)| \geq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at most

$$\exp\left(-\frac{\epsilon^4(1+\epsilon)^2(1-\epsilon)^{2(i-1)}\Delta^2}{2(\epsilon+\epsilon^2)(1-\epsilon)^{i-1}\Delta}\right) = \exp\left(-\frac{\epsilon^3(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{2}\right) \leq \exp\left(-\frac{1}{4}\epsilon^4\Delta\right) \leq \frac{1}{n^{25}}.$$

□

Lemma 10.1.14. *Let $u \in U$, $i \in [T]$. Then we have that $|F_i''(u)| \leq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at least $1 - 1/n^{33}$.*

Proof. Let $u \in U$, $i \in [T]$, and $u_1, \ldots, u_\ell$ be the nodes that are connected to u by an edge in $N_i(u)$. Arbitrarily fix all of the random bits used in the first $i-1$ rounds that determine the palettes $P_i(u), P_i(u_1), \ldots, P_i(u_\ell)$. Let $v \in \{u_1, \ldots, u_\ell\}$, $e = (u, v)$, and $v_1, \ldots, v_{\ell'}$ be the nodes that are connected to v by an edge in $N_i(u) \setminus \{e\}$. Recall that we can assume the input graph is $\mathcal{N}(u, T+1)$, which is a tree. Since the graph is a tree, the nodes $v, v_1, \ldots, v_\ell$ are all disconnected from each other in $(V, S_{<i})$, and hence $\{X^c_{v_1}\}_c, \ldots, \{X^c_{v_\ell}\}_c$ are mutually independent families of random variables by Lemmas 10.1.6 and 10.1.7.

Given some $f \in N_i(v) \setminus \{e\}$, the probability that e and f conflict is the probability that e and f are assigned the same tentative color. Letting $c = \tilde{\chi}(e)$, we have by Lemma 10.1.8 that $\Pr[\tilde{\chi}(f) = c] \leq 1/|P_i(v)|$. Let $f, f' \in N_i(v) \setminus \{e\}$ be distinct edges such that $f = (v, w)$ and $f = (v, w')$. Then the event $\tilde{\chi}(f) = c$ depends on the random bits that determine $P_i(w)$ and the random bits used to sample $\tilde{\chi}(f)$ (recall that the bits that determine $P_i(v)$ are fixed). Hence, by Lemma 10.1.6, the events $\tilde{\chi}(f) = c$ and $\tilde{\chi}(f') = c$ are independent since they depend on distinct random bits. It follows that

$$\Pr[e \notin F_i''(u)] = \prod_{f \in N_i(v)\setminus\{e\}} \Pr\left[\tilde{\chi}(f) \neq c\right] \geq \prod_{f \in N_i(v)\setminus\{e\}} \left(1 - \frac{1}{|P_i(v)|}\right)$$

$$\geq \left(1 - \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}\right)^{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta} \geq 1 - \frac{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta} = 1 - \epsilon,$$

where the last inequality follows from Bernoulli's inequality. By linearity of expectation, we get that

$$\mathbb{E}[|F_i''(u)|] = \sum_{e \in N_i(u)} \Pr[e \in F_i''(u)] \leq \epsilon|N_i(u)| \leq (\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta.$$

We now establish concentration. Given $e = (u, v), e' = (u, v') \in N_i(u)$, the fact that the events $e \in F_i''(u)$ and $e' \in F_i''(u)$ depend on disjoint collections of random bits follow the details above and the fact that the graph is a tree. Hence, these events are independent. It follows that we can apply a Chernoff bound.

$$\Pr[|F_i''(u)| \geq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta] \leq \exp\left(-\frac{1}{3}(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta\right) \leq \exp\left(-\frac{\epsilon^3}{6}(1+\epsilon)\Delta\right) \leq 1/n^{33}.$$

□

Lemma 10.1.15. *Let $u \in U$, $i \in [T]$. Then we have that $|F_i'''(u)| \leq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at least $1 - 2/n^9$.*

Proof. We begin by fixing all of the random bits used in the first $i-1$ rounds such that for all $e \in N^2(U)$ we have $|P_i(e)| > (1-\epsilon^2)(1-\epsilon)^{2(i-1)}\Delta$. Note that by Lemma 10.1.11 this event occurs with probability at least $1 - 1/n^9$. Given some edge $e \in N_i(u)$, we now have that

$$\Pr[e \in F'''(u)] = \left(1 - \frac{|P_i(e)|}{(1+\epsilon)\Delta}\right)^{8\log(1/\epsilon)/\epsilon^2} \leq \left(1 - (1-\epsilon)(1-\epsilon)^{2(i-1)}\right)^{8\log(1/\epsilon)/\epsilon^2}$$

$$\leq \left(1 - \epsilon^2/8\right)^{8\log(1/\epsilon)/\epsilon^2} \leq \epsilon,$$

where the first equality follows from the fact that we are drawing colors from $[(1+\epsilon)\Delta]$ independently and u.a.r while avoiding $P_i(e)$ and the second inequality follows from the fact that $(1-\epsilon)^{i-1} \geq \epsilon/2$. By linearity of expectation, it follows that

$$\mathbb{E}[|F_i'''(u)|] = \sum_{e \in N_i(u)} \Pr[e \in F_i'''(u)] \leq \epsilon |N_i(u)| < (\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta.$$

Given two distinct edges $e = (u, v), f = (u, w) \in N_i(u)$, the random bits used by the algorithm to sample the colors $\tilde{\chi}(e)$ and $\tilde{\chi}(f)$ are distinct, so we can apply Lemma 10.1.6 to get that the events $e \in F_i'''(u)$ and $f \in F_i'''(u)$ are independent. Hence, we can apply Chernoff bounds to get that

$$\Pr[|F_i'''(u)| \geq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta] \leq \exp\left(-\frac{1}{3}(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta\right) \leq \exp\left(-\frac{\epsilon^3}{6}(1+\epsilon)\Delta\right) \leq \frac{1}{n^{33}}.$$

□

Let G_F denote the subgraph of G consisting of the edges contained in F. We now remove the conditioning on event $\mathcal{Z}$.

Lemma 10.1.16. *We have that $\deg_{G_F}(u) \leq 7\epsilon(1+\epsilon)\Delta$ for all $u \in U$ with probability at least $1 - 6/n^7$.*

Proof. Let $u \in U$. Then, by Lemmas 10.1.12, 10.1.13, 10.1.14 and 10.1.15 we get that

$$\deg_{G_F}(u) = |F_{T+1}(u)| + \sum_{i=1}^{T} |F_i(u)| \leq |F_{T+1}(u)| + \sum_{i=1}^{T} \left(|F_i'(u)| + |F_i''(u)| + |F_i'''(u)|\right)$$

$$\leq \epsilon(1+\epsilon)\Delta + 6(\epsilon^2+\epsilon^3)\Delta \sum_{i=1}^{T}(1-\epsilon)^{i-1} = \epsilon(1+\epsilon)\Delta + 6(\epsilon^2+\epsilon^3)\Delta \cdot \frac{1-(1-\epsilon)^T}{\epsilon} \leq 7\epsilon(1+\epsilon)\Delta.$$

with probability at least $1 - 1/n^{33} - T/n^{25} - T/n^{33} - 2T/n^9 \geq 1 - 5/n^8$. The lemma follows by union bounding over all nodes in U and removing the conditioning on event $\mathcal{Z}$. □

10.1.3 Properties of Algorithm 21

Let $G = (V, E)$ be a graph with maximum degree at most Δ and $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ be the subgraphs obtained from calling Algorithm 21 on G with some $2 \leq \Delta' \leq \Delta$.

Lemma 10.1.17. *$\Delta(\mathcal{G}_j) \leq \Delta' + 10\sqrt{\Delta' \log n}$ for all $j \in [\eta]$ with probability at least $1 - 1/n^{31}$.*

Proof. Let $j \in [\eta]$ and $u \in V$. Let X_j^e be the indicator for the event that some edge $e \in E$ is contained in the graph $\mathcal{G}_j$. Then clearly $\deg_{\mathcal{G}_j}(u) = \sum_{e \in N(u)} X_j^e$, and hence

$$\mathbb{E}[\deg_{\mathcal{G}_j}(u)] = \sum_{e \in N_G(u)} \Pr[e \in \mathcal{E}_j] \leq \frac{\Delta}{\eta} \leq \Delta'.$$

By applying Chernoff bounds, we get that

$$\Pr\left[\deg_{\mathcal{G}_j}(u) > \Delta' + 10\sqrt{\Delta' \log n}\right] \leq \exp\left(-\frac{1}{3} \cdot \frac{100 \log n}{\Delta'} \cdot \Delta'\right) \leq \frac{1}{n^{33}},$$

which implies that $\deg_{\mathcal{G}_j}(u) \leq \Delta' + 10\sqrt{\Delta' \log n}$ with probability at least $1 - 1/n^{33}$. We can union bound over all $u \in V$ and $j \in [\eta]$ to get that $\Delta(\mathcal{G}_j) \leq \Delta' + 10\sqrt{\Delta' \log n}$ for all $j \in [\eta]$ with probability at least $1 - n\eta/n^{33} \geq 1 - 1/n^{31}$. □

The following lemma is proven by [KLS+22].

Lemma 10.1.18 (Lemma 4.2, [KLS+22])**.** *Let G' be a subgraph of G obtained by sampling each edge in G independently with probability D/Δ, where $D \geq 2$. Then the probability that the g-neighborhood of an edge e in G' contains a cycle is at most $3D^{5g}/\Delta$.*

We will use the following lemma, which follows immediately from Lemma 10.1.18.

Lemma 10.1.19. *Let G' be a subgraph of G obtained by sampling each edge in G independently with probability D/Δ, where $D \geq 2$. Then the probability that the g-neighborhood of a node u contains a cycle in G' is at most $3D^{5g}/\Delta$.*

Lemma 10.1.20. *Let $G^\star$ be the subgraph of G that contains an edge e iff $e \in \mathcal{E}_j$ is incident on a node u such that the g-neighborhood of e in $\mathcal{G}_j$ is not a tree. Then $\Delta(G^\star) \leq (\Delta' + 10\sqrt{\Delta' \log n}) \cdot (6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n})$ with probability at least $1 - 1/n^{30}$.*

Proof. Given some $u \in V$, $j \in [\eta]$, define X_j^u to be the indicator for the event that the $(g+1)$-neighborhood of u in the graph $\mathcal{G}_j$ is not a tree. It immediately follows that

$$\deg_{G^\star}(u) \leq \sum_{j \in [\eta]} X_j^u \cdot |N(u) \cap \mathcal{E}_j| \leq \sum_{j \in [\eta]} X_j^u \cdot \Delta(\mathcal{G}_j) \leq \max_{j \in [\eta]} \Delta(\mathcal{G}_j) \cdot \sum_{j \in [\eta]} X_j^u.$$

By Lemma 10.1.17, we have that $\max_j \Delta(\mathcal{G}_j) \leq \Delta' + 10\sqrt{\Delta' \log n}$ with probability at least $1 - 1/n^{31}$. It follows from Lemma 10.1.19 that the $g+1$-neighborhood of u in the graph $\mathcal{G}_j$ is not a tree with probability at most $3(\Delta/\eta)^{5(g+1)}/\Delta \leq 3(\Delta')^{5(g+1)}/\Delta$. Hence, letting X^u denote $\sum_{j \in [\eta]} X_j^u$,

$$\mathbb{E}[X^u] \leq \sum_{j \in [\eta]} \mathbb{E}[X_j^u] \leq \sum_{j \in [\eta]} 3(\Delta')^{5(g+1)}/\Delta \leq 6(\Delta')^{5(g+1)}$$

holds for all $u \in V$. In order to establish concentration, we first establish that, for any fixed u, the random variables $\{X_j^u\}_j$ are NA (see Definition 10.5.5). Given some $e \in E$, let X_j^e indicate the event that $e \in \mathcal{E}_j$. For any fixed j, the random variables $\{X_j^e\}_j$ are NA by Proposition 10.5.6. Since the families of random variables $\{X_j^{e_1}\}_j, \ldots, \{X_j^{e_m}\}_j$ are mutually independent, it follows by closure under products (Proposition 10.5.9) that $\{X_j^e\}_{j,e}$ are NA. Finally, for any $u \in V$, since X_j^u is a monotonically increasing function of $X_j^{e_1}, \ldots, X_j^{e_m}$, it follows by disjoint monotone aggregation (Proposition 10.5.9) that $\{X_j^u\}_j$ are NA. Hence, we can apply Hoeffding bounds for NA random variables to get that

$$\Pr\left[X^u > 6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n}\right] \leq \exp\left(-2 \cdot \frac{100(\Delta/\Delta')\log n}{\eta}\right) \leq \frac{1}{n^{100}}.$$

By union bounding over all $u \in V$, it follows that, with probability at least $1 - 1/n^{99}$,

$$X^u \leq 6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n}$$

for all $u \in V$. Putting everything together and applying a union bound we get that with probability at least $1 - 1/n^{30}$

$$\Delta(G^\star) \leq \left(\Delta' + 10\sqrt{\Delta' \log n}\right) \cdot \left(6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n}\right). \qquad \square$$

10.1.4 Edge Coloring the Subsampled Graphs

Let $G = (V, E)$ be a graph with maximum degree at most Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{30T}$. Let $\gamma = 1/(30T)$ and $\Delta' = \Delta^\gamma$. Now suppose we run Algorithm 20 on input (G, Δ, ϵ). Let $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ denote the partition of G produced by the algorithm. Recall that, for $j \in [\eta]$, $\mathcal{F}_j$ denotes the set of failed edges in $\mathcal{G}_j$. Let $\mathcal{F} = \bigcup_{j \in [\eta]} \mathcal{F}_j$. For notational convenience, we identify $\mathcal{F}$ with the graph

$(V, \mathcal{F})$ for the rest of this section. Finally, let $G^\star$ denote the graph that contains an edge $e \in \mathcal{E}_j$ if and only if one of the endpoints of e is bad with respect to $\mathcal{G}_j$. Then we have the total number of colors used by our algorithm is

$$\sum_{j \in [\eta]} (1+\epsilon)^2 \Delta' + 3\Delta(\mathcal{F}),$$

where the factor of 3 in the second term comes from the greedy algorithm. We now proceed to show that, with high probability, this expression is upper bounded by $(1 + O(\epsilon))\Delta$. We begin with the following simple observation.

Observation 10.1.21. *For all $j \in [\eta]$, $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta'$ with probability at least $1 - 1/n^{31}$.*

Proof. By Lemma 10.1.17, we have that for all $j \in [\eta]$

$$\Delta(\mathcal{G}_j) - \Delta' \leq 10\sqrt{\Delta' \log n} \leq 10\epsilon^2 \Delta'/(10\sqrt{2}) \leq \epsilon \Delta'$$

with probability at least $1 - 1/n^{31}$. □

For the rest of this section, we assume that the event in the statement of Observation 10.1.21 occurs and implicitly condition all probabilities on this event unless explicitly stated otherwise. Note that we also have $(1+\epsilon)\Delta' \geq (100 \log n)/\epsilon^2$.

Lemma 10.1.22. *We have that $\sum_{j \in [\eta]} (1+\epsilon)^2 \Delta' \leq (1+4\epsilon)\Delta$.*

Proof.

$$\sum_{j \in [\eta]} (1+\epsilon)^2 \Delta' \leq (1+\epsilon)^2 \Delta' \eta \leq (1+\epsilon)^2 \Delta + (1+\epsilon)^2 \leq (1+4\epsilon)\Delta.$$ □

Lemma 10.1.23. *Let $u \in V$ and let $J_u^\star = \{j \in [\eta] \,|\, u \text{ is good with respect to } \mathcal{G}_j\}$. Then we have that*

$$\deg_{\mathcal{F}}(u) \leq \Delta(G^\star) + \sum_{j \in J_u^\star} \deg_{\mathcal{F}_j}(u)$$

Proof. We have that

$$\deg_{\mathcal{F}}(u) = \sum_{j \in [\eta]} \deg_{\mathcal{F}_j}(u) = \sum_{j \in J_u^\star} \deg_{\mathcal{F}_j}(u) + \sum_{j \in [\eta] \setminus J_u^\star} \deg_{\mathcal{F}_j}(u).$$

By the definition of the graph $G^\star$, for any $j \in [\eta] \setminus J_u^\star$, all of the edges incident on u in $\mathcal{G}_j$ are contained in $G^\star$. Since the $\mathcal{G}_j$ are edge-disjoint, it follows that

$$\sum_{j \in [\eta] \setminus J_u^\star} \deg_{\mathcal{F}_j}(u) \leq \sum_{j \in [\eta] \setminus J_u^\star} \deg_{\mathcal{G}_j}(u) \leq \deg_{G^\star}(u) \leq \Delta(G^\star).$$ □

Lemma 10.1.24. *For all $j \in [\eta]$, $\deg_{\mathcal{F}_j}(u) \leq 9\epsilon\Delta'$ for all $u \in V$ such that u is good with respect to $\mathcal{G}_j$ with probability at least $1 - 6/n^6$.*

Proof. Let $j \in [\eta]$. It follows from Lemma 10.1.16 that, for all $u \in V$ that are good with respect to $\mathcal{G}_j$, $\deg_{\mathcal{F}_j}(u) \leq 7\epsilon(1+\epsilon)^2\Delta' \leq 9\epsilon\Delta'$ with probability at least $1 - 6/n^7$. The result follows by union bounding over all $j \in [\eta]$. □

Lemma 10.1.25. *We have that $\Delta(G^\star) \leq \epsilon\Delta$ with probability at least $1 - 1/n^{30}$.*

Proof. By Lemma 10.1.20, with probability at least $1 - 1/n^{30}$ we have that

$$\begin{aligned}
\Delta(G^\star) &\leq \left(\Delta^\gamma + \Delta^{\gamma/2} \cdot 10\sqrt{\log n}\right) \cdot \left(\Delta^{5\gamma(T+2)} \cdot 6 + \Delta^{(1-\gamma)/2} \cdot 10\sqrt{\log n}\right) \\
&\leq \Delta^{5\gamma T+11\gamma} \cdot 6 + \Delta^{(1+\gamma)/2} \cdot 10\sqrt{\log n} + \Delta^{5\gamma T+11\gamma} \cdot 60\sqrt{\log n} + \Delta^{1/2} \cdot 100 \log n \\
&\leq 100 \log n \cdot \left(\Delta^{5\gamma T+11\gamma} + \Delta^{(1+\gamma)/2} + \Delta^{1/2}\right) \\
&\leq \left(\Delta^{5/30+11/(30T)} + \Delta^{1/2+1/(60T)}\right) \cdot 200 \log n \\
&\leq \left(\Delta^{5/30+11/(30T)} + \Delta^{1/2+1/(60T)}\right) \cdot 300\epsilon^4\Delta^{1/(30T)}/200 \\
&\leq 4\epsilon^4\Delta \\
&\leq \epsilon\Delta.
\end{aligned}$$

□

Lemma 10.1.26. *We have that $\Delta(\mathcal{F}) \leq 19\epsilon\Delta$ with probability at least $1 - 7/n^6$.*

Proof. It follows from the preceding lemmas that for all $u \in V$ we have that

$$\deg_{\mathcal{F}}(u) \leq \Delta(G^\star) + \sum_{j \in J_u^\star} \deg_{\mathcal{F}_j}(u) \leq \epsilon\Delta + |J_u^\star| \cdot 9\epsilon\Delta' \leq 19\epsilon\Delta$$

with probability at least $1 - 1/n^{30} - 6/n^6 \geq 1 - 7/n^6$. It immediately follows that $\Delta(\mathcal{F}) \leq 19\epsilon\Delta$ with the same probability. □

We are not ready to complete the proof of Theorem 10.1.2.

Proof of Theorem 10.1.2. First, observe that our algorithm uses at most

$$\sum_{i \in [\eta]} (1+\epsilon)^2\Delta' + 3\Delta(\mathcal{F})$$

colors. It now follows from Lemmas 10.1.22 and 10.1.26 that

$$\sum_{i \in [\eta]} (1+\epsilon)^2\Delta' + 3\Delta(\mathcal{F}) \leq (1+4\epsilon)\Delta + 3 \cdot 19\epsilon\Delta = (1+61\epsilon)\Delta$$

with probability at least $1 - 1/n^{31} - 7/n^6 \geq 1 - 8/n^6$, where the terms in the probability come from removing the conditioning on the event from Observation 10.1.21 and the probability in Lemma 10.1.26. □

10.2 Our Dynamic Algorithm with Constant Recourse

In this section, we describe our dynamic algorithm. For now, we omit all details of implementation and deal only with bounding the *recourse* of our dynamic algorithm. We design the data structures that allow us to implement this procedure efficiently in the proceeding sections. The main result in this section is the following theorem.

Theorem 10.2.1. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1+61\epsilon)\Delta$-edge coloring of G and has an expected worst-case recourse of $O(1/\epsilon^4)$.*

The Dynamic Setting. In the dynamic setting, we have a graph $G = (V, E)$ that undergoes updates via a sequence of edge insertions and deletions, while the set of nodes remains fixed. Our task is to explicitly maintain an edge coloring χ of G as it is updated, where Δ is a fixed upper bound on the maximum degree of the graph of G at any point. Let $\sigma_1, ..., \sigma_\kappa$ denote the sequence of updates, and $G^{(t)} = (V, E^{(t)})$ denote the state of the graph G after the first t updates. We assume that the graph G is initially empty, i.e. $G^{(0)} = (V, \emptyset)$. Given some dynamic edge coloring algorithm, its *update time* is the time it takes to handle an update, and its *recourse* is the number of edges that change color during an update. In this chapter, we assume that all adversaries are *oblivious*. In other words, the update σ_t does not depend on the random bits used by our algorithm while handling the updates $\sigma_1, \ldots, \sigma_{t-1}$.

Our Algorithm. Informally, our dynamic algorithm works by maintaining the output of StaticColor on the dynamic graph G as it undergoes edge insertions and deletions. In other words, we maintain the invariant that at any point in time the coloring generated by our dynamic algorithm on the current input graph G is the same as the coloring obtained by running StaticColor on G—regardless of what updates have occurred previously. Since we fix the randomness for every potential edge in advance, the tentative colors assigned by StaticColor to the edges in phase 2 (while running Algorithm 22 on the subgraphs $\mathcal{G}_j$) are completely determined by which edges are present in G. Hence, the tentative color $\tilde{\chi}^{(t)}(e)$ of each edge $e \in E^{(t)}$ is well defined. It follows from Theorem 10.1.2 that as long as $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, given some $t \in [\kappa]$, the edge coloring $\chi^{(t)}$ (defined by combining the tentative colorings $\tilde{\chi}^{(t)}$ for the edges in each $\mathcal{G}_j$ that don't fail and the greedy coloring of H) uses at most $(1+61\epsilon)\Delta$ colors with probability at least $1 - O(1/n^6)$.

We design data structures that allow us to maintain these tentative colorings explicitly, along with the sets of edges that fail, for each graph $\mathcal{G}_j$ as edges are inserted and deleted from G. We

then dynamically maintain a greedy edge coloring of the graph H that uses $3\Delta(H)$ many colors and only changes the colors of $O(1)$ many edges every time an edge is inserted or deleted from H. This level of detail is sufficient to upper bound the recourse of our dynamic algorithm. We begin by bounding the recourse of our algorithm and defer the details of how to implement this efficiently to the proceeding sections.

10.2.1 Recourse analysis

We now proceed to upper bound the expected recourse of our algorithm. We achieve this by arguing that in expectation the tentative colorings produced by StaticColor on two graphs that differ only by one edge assign different colors to at most $O(1/\epsilon^4)$ many edges. By showing that the number of edges that are added and removed from H is at most a constant factor larger than the number of edges that change their tentative colors, it then follows that the expected recourse of our algorithm is $O(1/\epsilon^4)$.

Let $t \in [\kappa]$. Suppose that the t^{th} update corresponds to the insertion or deletion of an edge e contained in $\mathcal{G}_j$. Then clearly only edges contained in $\mathcal{G}_j$ can change their tentative colors during this update. Since the colors used by the tentative colorings in each of $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ are distinct, it also follows that only edges in $\mathcal{G}_j$ can be added or removed from H during the update. It follows that while bounding the recourse of the t^{th} update it is sufficient to only consider the edges contained in the graph $\mathcal{G}_j$. It is not too difficult to see that the recourse of the t^{th} update is upper bounded by

$$O\left(\left|\mathcal{F}_j^{(t-1)} \oplus \mathcal{F}_j^{(t)}\right|\right) + \left|\left\{e \in \mathcal{E}_j^{(t)} \;\middle|\; \tilde{\chi}^{(t-1)}(e) \neq \tilde{\chi}^{(t)}(e)\right\}\right|,$$

where the terms correspond to the changes in the greedy coloring caused by edge insertions and deletions from H due to edges that failed at time $t-1$ but not at time t and vice versa, and the edges that change their tentative colors, respectively.

Notation. In order to emphasize that we are looking at the state of an object X *directly after the* t^{th} *update* we add the superscript $X^{(t)}$. For example, $P_i^{(t)}(e)$ is the set $P_i(e)$ after the t^{th} update, i.e. when the input graph to our algorithm is $G^{(t)}$, where $P_i(e)$ is the palette of the edge e during round i as defined in Section 10.1. Since while bounding the recourse of the t^{th} update it is sufficient to only consider the edges contained in the graph $\mathcal{G}_j$, when we use notation from Algorithm 22, it is implicit that the notation is referring to objects from the call to this algorithm on the graph $\mathcal{G}_j$. We will now introduce some definitions that will allow us to analyze the way that changes in the tentative coloring propagate through the rounds of Algorithm 22 after an update.

The Sets $\Gamma(e)$ and $\Lambda(e)$. The following definitions capture the notion of an edge e having an effect on the tentative color assigned to some edge f in a subsequent round and are crucial for efficiently identifying the edges whose tentative colors change after an update.

Definition 10.2.2. *Given some $t \in [\kappa]$, $i \in [T]$, and an edge $e \in S_i^{(t-1)} \cup S_i^{(t)}$, we define the sets of*

edges $\Gamma^{(t)}(e)$ *and* $\Lambda^{(t)}(e)$ *by*

$$\Gamma^{(t)}(e) := \left\{ f \in N_{>i}^{(t)}(e) \;\middle|\; \tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e) \right\},$$

$$\Lambda^{(t)}(e) := \left\{ f \in N_{>i}^{(t)}(e) \;\middle|\; \tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t-1)}(e) \right\}.$$

We denote the sets $S_i^{(t)} \cap \Gamma^{(t)}(e)$ and $S_i^{(t)} \cap \Lambda^{(t)}(e)$ by $\Gamma_i^{(t)}(e)$ and $\Lambda_i^{(t)}(e)$ respectively. Informally, one can think of the edges in $\Gamma^{(t)}(e)$ as the edges that change their color during the t^{th} update because e now occupies their color, and the edges in $\Lambda^{(t)}(e)$ as the edges that change their color during the t^{th} update because e no longer occupies a color that they would rather have assigned to them.

Type A and B Dirty Edges. For some $t \in [\kappa]$, $t \in [T]$, we define the sets of edges $A_i^{(t)}$ and $B_i^{(t)}$ by

$$A_i^{(t)} := \left\{ e \in S_i^{(t-1)} \cup S_i^{(t)} \;\middle|\; \tilde{\chi}^{(t-1)}(e) \neq \tilde{\chi}^{(t)}(e) \right\},$$

$$B_i^{(t)} := F_i^{(t-1)} \oplus F_i^{(t)}.$$

In words, $A_i^{(t)}$ is the set of edges in round i that change their tentative color during the t^{th} update, and $B_i^{(t)}$ is the set of edges in round i that fail at either time $t-1$ or t, but not both. We let $A^{(t)}$ and $B^{(t)}$ denote the sets $\bigcup_i A_i^{(t)}$ and $\bigcup_i B_i^{(t)}$ respectively, and refer to the edges in $A^{(t)}$ and $B^{(t)}$ as A-dirty and B-dirty respectively. We define an edge e as being *dirty* with respect to the t^{th} update if the color $\chi(e)$ changes during the t^{th} update and denote the set of such edges by $D^{(t)}$, and $D^{(t)} \cap S_i^{(t)}$ by $D_i^{(t)}$. The recourse of the t^{th} update is precisely $|D^{(t)}|$.

Basic Facts

We now give some basic facts about these definitions. Let $t \in [\kappa]$, $e^\star$ be the edge that is either inserted or deleted during the t^{th} update, and let $i^\star$ denote $i_{e^\star}$.

Lemma 10.2.3. *We have that* $|D^{(t)}| \leq O(|B^{(t)}|) + |A^{(t)}|$.

Proof. It is sufficient to argue that at most $O(|B^{(t)}|)$ many edges not in $A^{(t)}$ change their colors during the t^{th} update. Clearly, any such edge must be contained in $F^{(t-1)} \cup F^{(t)}$. Since at most $O(|F^{(t-1)} \oplus F^{(t)}|)$ many edges in $F^{(t-1)} \cup F^{(t)}$ change their colors during the t^{th} update (recall that our dynamic greedy algorithm changes the colors of at most $O(1)$ many edges in H when adding or removing an edge from H) the lemma follows. □

Lemma 10.2.4. *We have that* $|B^{(t)}| \leq 4|A^{(t)}| + 1$.

Proof. Consider the set of all the failed edges

$$F^{(t)} = \left\{ e \in E^{(t)} \;\middle|\; \exists f \in N_{i_e}^{(t)}(e) \text{ such that } \tilde{\chi}^{(t)}(e) = \tilde{\chi}^{(t)}(f) \right\} \cup \left\{ e \in E^{(t)} \;\middle|\; \tilde{\chi}^{(t)}(e) = \perp \right\}$$

that is defined by our tentative coloring $\tilde{\chi}^{(t)}$. We want to get an upper bound on the size of $B^{(t)} = F^{(t-1)} \oplus F^{(t)}$. Suppose we start with the set $F^{(t-1)}$ and are given the set of edges that

change their tentative colors during the t^{th} update, $A^{(t)}$. Let $A^{(t)} = \{e_1, \ldots, e_\ell\}$. Now suppose we update the tentative colors of the first r edges $e_1, \ldots, e_r \in A^{(t)}$, and let $F(r)$ be the set of failed edges defined with respect to the tentative coloring where an edge $f \in E^{(t-1)} \cup E^{(t)} \setminus \{e_1, \ldots, e_r\}$ receives color $\tilde{\chi}^{(t-1)}(f)$ and the edges $e_1, \ldots, e_r$ receive colors $\tilde{\chi}^{(t)}(e_1), \ldots, \tilde{\chi}^{(t)}(e_r)$ respectively. We can see that

$$|F^{(t-1)} \oplus F^{(t)}| \leq |F^{(t-1)} \oplus F(0)| + |F(0) \oplus F(1)| + |F(1) \oplus F(2)| + \cdots + |F(\ell-1) \oplus F(\ell)|.$$

Let $r \in [\ell]$ and let $e_r = (u, v)$, and consider how $F(r-1)$ changes into $F(r)$ after we change the tentative color of e_r from $\tilde{\chi}^{(t-1)}(e_r)$ to $\tilde{\chi}^{(t)}(e_r)$ (assume for now that neither color is $\perp$). We have that $|F(r) \setminus F(r-1)| \leq 2$. This is because the only edges in $F(r) \setminus F(r-1)$ are edges incident to u and v with color $\tilde{\chi}^{(t)}(e_r)$ that are not already in $F(r-1)$, and there can be at most one such edge incident on each of u and v. By an analogous argument, $|F(r-1) \setminus F(r)| \leq 2$. It follows that $|F(r-1) \oplus F(r)| \leq 4$. A similar argument shows that if one of these colors is $\perp$ then $|F(r-1) \oplus F(r)| \leq 3$. Finally, we note that $F^{(t-1)} \oplus F(0) \subseteq \{e^\star\}$ and the lemma follows. □

Corollary 10.2.5. *We have that* $|D^{(t)}| \leq O(|A^{(t)}|) + O(1)$.

Proof. Follows immediately from Lemmas 10.2.3 and 10.2.4. □

Lemma 10.2.6. *For all* $e \notin A^{(t)}$, *we have that* $\Gamma^{(t)}(e) = \emptyset$ *and* $\Lambda^{(t)}(e) = \emptyset$.

Proof. Since $e \notin A^{(t)}$, we have that $\tilde{\chi}^{(t)}(e) = \tilde{\chi}^{(t-1)}(e)$. Hence, $\Gamma^{(t)}(e) = \{f \in N^{(t)}_{<i_e}(e) \,|\, \tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t-1)}(e)\}$ which is clearly empty since, for any $f \in \Gamma^{(t)}(e)$, e and f share an endpoint and $i_e < i_f$, and hence cannot have the same tentative color. Similarly, the set $\Lambda^{(t)}(e)$ is $\{f \in N^{(t)}_{>i_e}(e) \,|\, \tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t)}(e)\}$ and is also empty by the same arguments. □

Lemma 10.2.7. *For all* $e \in E^{(t-1)} \cup E^{(t)}$, *we have that* $\Gamma^{(t)}(e) \subseteq A^{(t)}$ *and* $\Lambda^{(t)}(e) \subseteq A^{(t)}$.

Proof. Let $e \in E^{(t)} \cup E^{(t-1)}$ and $f \in \Gamma^{(t)}(e)$. Then we know that $\tilde{\chi}^{(t)}(e) = \tilde{\chi}^{(t-1)}(f)$. Since e and f share an endpoint and $i_e < i_f$, we also have that $\tilde{\chi}^{(t)}(e) \neq \tilde{\chi}^{(t)}(f)$. Hence, it follows that $\tilde{\chi}^{(t)}(f) \neq \tilde{\chi}^{(t-1)}(f)$ and so $f \in A^{(t)}$. It follows that $\Gamma^{(t)}(e) \subseteq A^{(t)}$. By an analogous argument, we have that $\Lambda^{(t)}(e) \subseteq A^{(t)}$. □

Lemma 10.2.8. *For all* i *such that* $i^\star < i \leq T$, *we have that* $A^{(t)}_i \subseteq \Gamma^{(t)}(A^{(t)}_{<i}) \cup \Lambda^{(t)}(A^{(t)}_{<i})$.

Proof. We prove this lemma by showing that $\{e \in A^{(t)}_i \,|\, \ell^{(t)}_e > \ell^{(t-1)}_e\} \subseteq \Gamma^{(t)}(A^{(t)}_{<i})$ and $\{e \in A^{(t)}_i \,|\, \ell^{(t)}_e < \ell^{(t-1)}_e\} \subseteq \Lambda^{(t)}(A^{(t)}_{<i})$, which implies that

$$\begin{aligned} A^{(t)}_i &= \{e \in A^{(t)}_i \,|\, \ell^{(t)}_e > \ell^{(t-1)}_e\} \sqcup \{e \in A^{(t)}_i \,|\, \ell^{(t)}_e < \ell^{(t-1)}_e\} \\ &\subseteq \Gamma^{(t)}(A^{(t)}_{<i}) \cup \Lambda^{(t)}(A^{(t)}_{<i}). \end{aligned}$$

Let $e \in A_i^{(t)}$ such that $\ell_e^{(t)} > \ell_e^{(t-1)}$ and $i = i_e$. Then there exists some $f \in N^{(t)}(e) \cap S_{<i}^{(t)}$ such that $\tilde{\chi}^{(t)}(f) = c_e(\ell_e^{(t-1)}) = \tilde{\chi}^{(t-1)}(e)$. Hence, $e \in \Gamma^{(t)}(f)$. Since e and f share an endpoint and $i_f < i_e$, we must have that $\tilde{\chi}^{(t-1)}(f) \neq \tilde{\chi}^{(t-1)}(e) = \tilde{\chi}^{(t)}(f)$ and so $f \in A_{<i}^{(t)}$. It follows that $e \in \Gamma^{(t)}(A_{<i}^{(t)})$ and hence $\{e \in A_i^{(t)} \mid \ell_e^{(t)} > \ell_e^{(t-1)}\} \subseteq \Gamma^{(t)}(A_{<i}^{(t)})$. By a similar argument, we get that $\{e \in A_i^{(t)} \mid \ell_e^{(t)} < \ell_e^{(t-1)}\} \subseteq \Lambda^{(t)}(A_{<i}^{(t)})$. □

Corollary 10.2.9. *For all i such that $i^\star < i \leq T$, we have that $A_i^{(t)} = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)})$.*

Proof. It follows by Lemmas 10.2.7 and 10.2.8 that $A_i^{(t)} \subseteq \Gamma^{(t)}(A_{<i}^{(t)}) \cup \Lambda^{(t)}(A_{<i}^{(t)}) \subseteq A^{(t)}$. By intersecting with $S_i^{(t)}$, it follows that $A_i^{(t)} \subseteq \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)}) \subseteq A_i^{(t)}$. □

Bounding the Expected Recourse

We can now bound the expected recourse of our dynamic algorithm by showing that $\mathbb{E}[|A^{(t)}|] = O(1/\epsilon^4)$ for all $t \in [\kappa]$. By then applying Corollary 10.2.5, this immediately implies that the expected recourse of our algorithm while handling an update is $O(1/\epsilon^4)$. We devote the rest of this section to proving the following lemma.

Lemma 10.2.10. *For all $t \in [\kappa]$, $\mathbb{E}\left[\left|A^{(t)}\right|\right] \leq 1/\epsilon^4 + o(1)$.*

10.2.2 Proof of Lemma 10.2.10

For the remainder of this section, we fix some $t \in [\kappa]$. Let $e^\star$ denote the edge that is either inserted or deleted during the t^{th} update and let $i^\star$ denote $i_{e^\star}$. We can make the following observation.

Observation 10.2.11. *$A_i^{(t)} = \emptyset$ for all $0 \leq i < i^\star$ and $A_{i^\star}^{(t)} \subseteq \{e^\star\}$.*

We now establish a relationship between the expected sizes of the sets $A_i^{(t)}$ when i is larger than $i^\star$, and use this to bound the expected size of $A^{(t)}$.

Let $\mathcal{E}$ be the event that $\mathcal{N}(e^\star, 2T+2)$ is a tree at time t and $t-1$. Recall that since we are only concerned with bounding the recourse at time t, we only consider the edges in the graph $\mathcal{G}_j$, where $e^\star$ is contained in $\mathcal{G}_j$. Hence, $\mathcal{N}(e^\star, 2T+2)$ denotes $\mathcal{N}_{\mathcal{G}_j}(e^\star, 2T+2)$ in this context. Let $\mathcal{Z}^{(t)}$ denote the event that $\mathcal{Z}$ (defined in Section 10.1.2) occurs at time t. Let $\mathcal{Y}^{(t)}$ denote the event that the statement in Observation 10.1.21 occurs at time t. We first argue that we can assume that the event $\mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$ occurs.

Lemma 10.2.12. *We have that*

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \mid \mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}] + o(1).$$

Proof. By the law of total expectation, we have that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + \mathbb{E}[|A^{(t)}| \mid \neg\mathcal{E}] \cdot \Pr[\neg\mathcal{E}].$$

By Lemma 10.1.18, $\mathcal{E}$ does not occur with probability at most $3(\Delta')^{10T+10}/\Delta = 3\Delta^{1/(3T)-2/3}$. By Lemma 10.1.3, we can see that every edge in $A^{(t)}$ is contained in $\mathcal{N}(e^\star, T+1)$. Hence, the number of edges contained in $\mathcal{N}(e^\star, T+1)$ is an upper bound on $|A^{(t)}|$. By Observation 10.1.21, we have that $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta' = (1+\epsilon)\Delta^{1/(30T)}$ with probability at least $1-1/n^{31}$. It follows that $\mathcal{N}(e^\star, T+1)$ contains at most $2((1+\epsilon)\Delta^{1/(30T)})^{T+1} \leq (3/\epsilon)\Delta^{1/15}$ many edges with the same probability. Putting everything together, we get that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E}] \cdot \Pr[\mathcal{E}] + \mathbb{E}[|A^{(t)}| \,|\, \neg\mathcal{E}] \cdot \Pr[\neg\mathcal{E}] \leq \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E}] + 9\Delta^{-4/15}/\epsilon^2$$

with probability at least $1 - 1/n^{31}$. Noting that $|A^{(t)}| \leq |E| \leq n^2$, we have that

$$\mathbb{E}[|A^{(t)}|] \leq n^2 \cdot (1/n^{31}) + \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E}] + 9\Delta^{-4/15}/\epsilon^2 \leq \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E}] + 9\Delta^{-4/15}/\epsilon^2 + 1/n^{29}.$$

Finally, letting $\mathcal{X} = \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$, the result follows by noting that

$$\begin{aligned}\mathbb{E}[|A^{(t)}| \,|\, \mathcal{E}] &\leq \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E} \cap \mathcal{X}] + n^2 \cdot \Pr[\neg\mathcal{X}] \\ &\leq \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E} \cap \mathcal{X}] + 4/n^{12}.\end{aligned}$$

□

For the rest of this section, we assume that $\mathcal{N}(e^\star, 2T+2)$ is a tree at time t and $t-1$. We also fix the random bits used by the algorithm to determine the partition of the graph so that event $\mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$ occurs and the random bits used by the algorithm to determine the rounds of edges so that the event $\mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)}$ occurs. Note that, in this context, the upper bound on $\Delta(\mathcal{G}_j)$ is $(1+\epsilon)\Delta'$, where $\Delta' = \Delta^{1/(30T)}$. We implicitly condition all probabilities on these events unless stated otherwise. By Lemma 10.1.3, since all edges that change their tentative color or failed status during the t^{th} update are contained in $\mathcal{N}^{(t)}(e^\star, T+1)$, we can assume that the input graph is $\mathcal{N}^{(t)}(e^\star, 2T+2)$ while trying to bound the number of such edges.

Lemma 10.2.13. *Let $i \in [T]$ and $e \in S^{(t)}_{<i}$, then we have that $\mathbb{E}[|\Gamma^{(t)}_i(e)|] \leq 2\epsilon$.*

Proof. We begin by observing that by linearity of expectation

$$\mathbb{E}[|\Gamma^{(t)}_i(e)|] = \sum_{f \in N^{(t)}_i(e)} \Pr[\tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e)].$$

If $e \notin A^{(t)}_{<i}$, then by Lemma 10.2.6 we have that $\Gamma^{(t)}(e) = \emptyset$. Assume that $e \in A^{(t)}_{<i}$. Then e is contained in $\mathcal{N}(e^\star, T+1)$ and hence $\mathcal{N}(e, T+1) \subseteq \mathcal{N}(e^\star, 2T+2)$ is a tree. Let $e = (u, v)$ and fix the random bits used by the algorithm in the first $i-1$ rounds that determine the palettes $P^{(t-1)}_i(u)$ and $P^{(t-1)}_i(v)$. Let $f = (u, w) \in N^{(t-1)}_i(e)$ (note that $N^{(t)}_i(e) = N^{(t-1)}_i(e)$ since $i_{e^\star} < i$). Then e and w are disconnected in the graphs $(V, S^{(t-1)}_{<i})$ and $(V, S^{(t)}_{<i})$ since $\mathcal{N}^{(t)}(e^\star, 2T+2)$ and $\mathcal{N}^{(t-1)}(e^\star, 2T+2)$ are both trees and one is a subgraph of the other. Letting $c = \tilde{\chi}^{(t)}(e)$, we have

that c does not depend on the random bits that determine the palette $P_i^{(t-1)}(w)$. We can now apply Lemmas 10.1.8 and 10.1.10 to get that

$$\Pr[\tilde{\chi}^{(t-1)}(f) = c] \leq \frac{1}{|P_i^{(t-1)}(u)|} \leq \frac{1}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}.$$

The same holds for all $f \in N_i^{(t-1)}(v)$. It follows that

$$\mathbb{E}[|\Gamma_i^{(t)}(e)|] = \sum_{f \in N_i^{(t)}(e)} \Pr[\tilde{\chi}^{(t-1)}(f) = c]$$

$$\leq \sum_{f \in N_i^{(t-1)}(u)} \Pr[\tilde{\chi}^{(t-1)}(f) = c] + \sum_{f \in N_i^{(t-1)}(v)} \Pr[\tilde{\chi}^{(t-1)}(f) = c]$$

$$\leq \frac{|N_i^{(t-1)}(u)| + |N_i^{(t-1)}(v)|}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} \leq \frac{2\epsilon(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} = 2\epsilon. \qquad \square$$

Lemma 10.2.14. *Let $i \in [T]$ and $e \in S_{<i}^{(t)}$, then we have that $\mathbb{E}[|\Lambda_i^{(t)}(e)|] \leq 2\epsilon$.*

Proof. We begin by observing that by linearity of expectation

$$\mathbb{E}[|\Lambda_i^{(t)}(e)|] = \sum_{f \in N_i^{(t)}(e)} \Pr[\tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t-1)}(e)].$$

If $e \notin A_{<i}^{(t)}$, then by Lemma 10.2.6 we have that $\Lambda^{(t)}(e) = \emptyset$. Assume that $e \in A_{<i}^{(t)}$. Then e is contained in $\mathcal{N}(e^\star, T+1)$ and hence $\mathcal{N}(e, T+1) \subseteq \mathcal{N}(e^\star, 2T+2)$ is a tree. Let $e = (u, v)$ and fix the random bits used by the algorithm in the first $i-1$ rounds that determine the palettes $P_i^{(t)}(u)$ and $P_i^{(t)}(v)$. Let $f = (u, w) \in N_i^{(t)}(e)$. Then e and w are disconnected in the graphs $(V, S_{<i}^{(t-1)})$ and $(V, S_{<i}^{(t)})$ since $\mathcal{N}^{(t)}(e^\star, 2T+2)$ and $\mathcal{N}^{(t-1)}(e^\star, 2T+2)$ are both trees and one is a subgraph of the other. Letting $c = \tilde{\chi}^{(t)}(e)$, we have that c does not depend on the random bits that determine the palette $P_i^{(t)}(w)$. We can now apply Lemmas 10.1.8 and 10.1.10 to get that

$$\Pr[\tilde{\chi}^{(t)}(f) = c] \leq \frac{1}{|P_i^{(t)}(u)|} \leq \frac{1}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}.$$

The same holds for all $f \in N_i^{(t)}(v)$. It follows that

$$\mathbb{E}[|\Lambda_i^{(t)}(e)|] = \sum_{f \in N_i^{(t)}(e)} \Pr[\tilde{\chi}^{(t)}(f) = c]$$

$$\leq \sum_{f \in N_i^{(t)}(u)} \Pr[\tilde{\chi}^{(t)}(f) = c] + \sum_{f \in N_i^{(t)}(v)} \Pr[\tilde{\chi}^{(t)}(f) = c]$$

$$\leq \frac{|N_i^{(t)}(u)| + |N_i^{(t)}(v)|}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} \leq \frac{2\epsilon(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} = 2\epsilon. \qquad \square$$

Lemma 10.2.15. *For all i such that $i^\star < i \leq T$, we have that $\mathbb{E}[|A_i^{(t)}|] \leq 4\epsilon \cdot \mathbb{E}[|A_{<i}^{(t)}|]$.*

Proof. We know by Corollary 10.2.9 that

$$A_i^{(t)} = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)}) = \bigcup_{e \in A_{<i}^{(t)}} \left(\Gamma_i^{(t)}(e) \cup \Lambda_i^{(t)}(e)\right).$$

From this, we can immediately deduce that

$$\mathbb{E}[|A_i^{(t)}|] \leq \sum_{e \in A_{<i}^{(t)}} \left(\mathbb{E}[|\Gamma_i^{(t)}(e)|] + \mathbb{E}[|\Lambda_i^{(t)}(e)|]\right)$$

by using linearity of expectation. It then follows from Lemmas 10.2.13 and 10.2.14 that $\mathbb{E}[|A_i^{(t)}|] \leq 4\epsilon \cdot |A_{<i}^{(t)}|$. The lemma follows by taking expectations on both sides. $\square$

Lemma 10.2.16. *For all i such that $i^\star < i \leq T$, we have that $\mathbb{E}[|A_{\leq i}^{(t)}|] \leq (1+4\epsilon) \cdot \mathbb{E}[|A_{\leq i-1}^{(t)}|]$.*

Proof. By applying Lemma 10.2.15 we get that

$$\mathbb{E}[|A_{\leq i}^{(t)}|] = \mathbb{E}[|A_i^{(t)}|] + \mathbb{E}[|A_{<i}^{(t)}|] \leq 4\epsilon \cdot \mathbb{E}[|A_{<i}^{(t)}|] + \mathbb{E}[|A_{<i}^{(t)}|] = (1+4\epsilon) \cdot \mathbb{E}[|A_{\leq i-1}^{(t)}|]. \qquad \square$$

Lemma 10.2.17. *We have that $\mathbb{E}[|A^{(t)}|] \leq 1/\epsilon^4$.*

Proof. It follows from Observation 10.2.11 and Lemma 10.2.16 that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A_{\leq T}^{(t)}|] \leq (1+4\epsilon)^{T-i^\star} \cdot \mathbb{E}[|A_{\leq i^\star}^{(t)}|] \leq (1+4\epsilon)^T \leq e^{4\log(1/\epsilon)} = 1/\epsilon^4. \qquad \square$$

Finally, using Lemma 10.2.12, we remove the conditioning on the event $\mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$ to get that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \,|\, \mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)}] + o(1) = O(1/\epsilon^4).$$

10.3 Implementing our Dynamic Algorithm

We now proceed to give a more detailed description of our algorithm, which we then implement with appropriate data structures. The main result in this section is Corollary 10.3.5, which is restated below.

Theorem 10.3.1. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1+61\epsilon)\Delta$-edge coloring of G and has an expected worst-case update time of $O(\log^4(1/\epsilon)/\epsilon^9)$.*

10.3.1 Our Algorithm

High-Level Approach. Let $t \in [\kappa]$, and suppose we have the edge coloring $\chi^{(t-1)}$ of $G^{(t-1)}$. Let $e^\star$ be the edge inserted/deleted at time t and let $i^\star$ denote $i_{e^\star}$. We know that the edge $e^\star$ is A-dirty (as long as it doesn't fail to be assigned a tentative color) and that there are no other A-dirty edges in the first $i^\star$ rounds. The main idea behind our dynamic algorithm is to iterate through the rounds $i^\star + 1, \ldots, T$ and observe how the changes in the tentative coloring $\tilde{\chi}$ propagate through the rounds, finding all of the A-dirty edges. By designing an appropriate data structure that updates the failed edges as we update the tentative colors of edges, we show that we can explicitly maintain the failed edges by just finding the A-dirty edges and updating their tentative colors. We find all of the A-dirty edges in round i inductively by using Corollary 10.2.9. In other words, if we can find all of the A-dirty edges in the first $i-1$ rounds then we can find all of the A-dirty edges in round i by using the fact that $A_i^{(t)} = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)})$. Algorithm 23 gives the pseudocode for this high-level approach.

Algorithm 23: UPDATE-COLORING($e^\star$)

```
Find χ̃^(t)(e*)
if χ̃^(t)(e*) = χ̃^(t−1)(e*) then
    return
A_{i*}^(t) ← {e*}
for i = i* + 1, ..., T do
    A_i^(t) ← Γ_i^(t)(A_{<i}^(t)) ∪ Λ_i^(t)(A_{<i}^(t))
    Find χ̃^(t)(e) for all e ∈ A_i^(t)
```

Maintaining the Tentative Colorings. The main technical challenge that we face is maintaining the tentative colorings for the graphs $\mathcal{G}_j$. In Section 10.3.2, we design a data structure that, given a dynamic graph G', is capable of maintaining and updating a tentative coloring of G' so that the output matches that of Algorithm 22 when they use the same random bits. More precisely, our dynamic data structure will be given the round i_e and color sequence c_e for each edge e in G', and will maintain the color indices ℓ_e of each edge so that they match the corresponding indices produced by Algorithm 22 when run on input G'. This defines a tentative coloring (by setting $\tilde{\chi}(e) = c_e(\ell_e)$) and a corresponding set of failed edges, which the data structure maintains explicitly. Given an edge e to be inserted or deleted from G', our data structure is capable of efficiently identifying all of the edges that need to change their color indices in order for the values maintained by the data structure to match the output produced by running Algorithm 22 on the updated graph. We will create η copies $\mathcal{D}_1, \ldots, \mathcal{D}_\eta$ of this data structure and use them to maintain the tentative colorings of the graphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ by feeding the graph $\mathcal{G}_j$ to $\mathcal{D}_j$. Since each $\mathcal{D}_j$ explicitly maintains the edges that fail while tentatively coloring $\mathcal{G}_j$, we can use them to explicitly maintain the graph H consisting of all failed edges by adding or removing an edge e from H every time one of the $\mathcal{D}_j$ adds or removes e from its collection of failed edges. In Section 10.3.3, we prove the following lemma.

Lemma 10.3.2. *There exists a dynamic data structure that, given a dynamic graph G and a pa-*

rameter Δ, is capable of explicitly maintaining the tentative coloring $\tilde{\chi}$ and the set of failed edges F defined by running Algorithm 22 on G, and has an expected update time of $O(\log^4(1/\epsilon)/\epsilon^5 \cdot \mathbb{E}[|A|])$, where A is the set of A-dirty edges during an update.

Maintaining the Greedy Coloring. We now show that there exists an algorithm that can dynamically maintain a greedy coloring of a graph and has $O(1)$ expected update time. Our algorithm has the property that each edge insertion or deletion can only change the colors of at most $O(1)$ many edges in the graph. Furthermore, this coloring is maintained explicitly and hence can be queried in $O(1)$ time. More precisely, in Section 10.3.4, we prove the following lemma.

Lemma 10.3.3. *There exists a dynamic data structure that, given a dynamic graph $G = (V, E)$ that undergoes edge insertions and deletions, can explicitly maintain a $3\Delta(G)$-edge coloring of G, where $\Delta(G)$ is the current maximum degree of G, has an expected update time of $O(1)$, and only changes the colors of $O(1)$ many edges per update.*

Putting Everything Together. We can now use the dynamic algorithm that is described in Lemma 10.3.2 to maintain the tentative coloring $\tilde{\chi}$ and the graph H defined by running Algorithm 20 on G, and then run the dynamic greedy algorithm described in Lemma 10.3.3 on H in order to efficiently maintain a $3\Delta(H)$-edge coloring of H. Recall that we create η copies of this data structure and use them to maintain the tentative colorings and failed edges of each $\mathcal{G}_j$ individually. By Lemma 10.2.10, we know that the expected type A recourse (in the subsampled graph $\mathcal{G}_j$ that the update occurs) is $O(1/\epsilon^4)$, and hence the expected update time of our algorithm is $O(\log^4(1/\epsilon)/\epsilon^9)$. Since the graph H is maintained explicitly, we insert or delete at most $O(\log^4(1/\epsilon)/\epsilon^9)$ many edges from H. Since our dynamic greedy algorithm has an expected update time of $O(1)$, this only leads to a $O(1)$ multiplicative factor in overhead to the update time. Finally, we note that the assumption that the randomness is generated in advance was made purely for analytic purposes so that we could argue that our dynamic algorithm and our static algorithm produce the same output when using the same random bits. However, if we generate the random bits for an edge e on the fly when it is inserted into the graph (i.e. decide which graph $\mathcal{G}_j$ to place it into and sample its round i_e and color sequence c_e independently of all previous random processes) then the distribution of the random bits assigned to the edges in the graph at any given time does not change, and the same guarantees hold. Our main theorem follows.

Theorem 10.3.4. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1+61\epsilon)\Delta$-edge coloring of G with probability at least $1 - 8/n^6$ at each time $t \in [\kappa]$, and has an expected worst-case update time of $O(\log^4(1/\epsilon)/\epsilon^9)$.*

By our proof of Theorem 10.1.2, it follows that, at any point in time, the graph H has a maximum degree of at most $19\epsilon\Delta$ with probability at least $1-8/n^6$. Hence, in the event that $\Delta(H) > 19\epsilon\Delta$, we

can resample all of the randomness used by our algorithm by deleting and reinserting every edge (and resampling all of the random bits for each edge in the process). This will take $O_\epsilon(m)$ expected time, and we will have that $\Delta(H) > 19\epsilon\Delta$ with probability at most $8/n^6$ independently of the randomness that our algorithm had used before. We repeat this process of resampling all of the randomness until we have that $\Delta(H) \leq 19\epsilon\Delta$. In expectation, we do this at most $1/(1-8/n^6) - 1 = 1/\Omega(n^6)$ many times. Since we spend $O_\epsilon(m) \leq O_\epsilon(n^2)$ expected time resampling all the randomness each time we do this, it follows that this process takes $o(1)$ time in expectation. However, since our algorithm uses at most $(1+61\epsilon)\Delta$ colors in total as long as $\Delta(H) \leq 19\epsilon\Delta$, this ensures that we always maintain a $(1+61\epsilon)\Delta$-edge coloring of G, while only incurring an additive $o(1)$ factor in the expected worst-case update time. Thus, we have the following corollary.

Corollary 10.3.5. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1+61\epsilon)\Delta$-edge coloring of G and has an expected worst-case update time of $O(\log^4(1/\epsilon)/\epsilon^9)$.*

Finally, by amortizing over a sufficiently long sequence of updates and periodically resampling the random bits, we can get $O_\epsilon(1)$ amortized update time with high probability.

Corollary 10.3.6. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1+61\epsilon)\Delta$-edge coloring of G and has an amortized update time of $O(\log^4(1/\epsilon)/\epsilon^9)$ over the whole update sequence w.h.p. as long as κ is a sufficiently large polynomial in n.*

Proof. We first make a further modification to the algorithm from Corollary 10.3.5 so that, every n^2 updates, our algorithm samples new random bits for each edge in the same way as described above, taking $O_\epsilon(m)$ time in expectation. This splits up the update sequence $\sigma_1, \ldots, \sigma_\kappa$ in *epochs* of length n^2. Let X_i be a random variable denoting the total update time of our algorithm during the i^{th} epoch. We have that $\mathbb{E}[X_i] = O(n^2 \log^4(1/\epsilon)/\epsilon^9)$. Now note that, given the data structures that we use to implement our algorithm, there exists some polynomial $p(n)$ such that X_i is always at most $p(n)$. Let $X = \sum_i X_i$ be the total update time of our algorithm. Since we sample fresh randomness at the end of each epoch, the random variables $X_1, \ldots, X_\tau$ are independent. Hence, we can apply Hoeffding bounds to get that $X = O(\kappa \log^4(1/\epsilon)/\epsilon^9)$ with probability at least $1 - \exp\left(-\tau \cdot \Theta_\epsilon(n^4)/p(n)^2\right)$. Hence, as long as $\tau = \Theta(\kappa/n^2)$ is a sufficiently large polynomial in n, $X = O(\kappa \log^4(1/\epsilon)/\epsilon^9)$ and the amortized update time of our algorithm is $O(\log^4(1/\epsilon)/\epsilon^9)$ with high probability. □

The rest of this section is now devoted to proving Lemmas 10.3.2 and 10.3.3. In Section 10.3.2, we design the key data structure that we need to maintain a tentative coloring and associated failed

edges. In Section 10.3.3, we show how to use this data structure in order to achieve this and prove Lemma 10.3.2. Finally, in Section 10.3.4, we prove a (slight generalization of) Lemma 10.3.3.

10.3.2 Key Data Structure

In this section, we design a dynamic data structure that will allow us to dynamically maintain a tentative coloring and the corresponding set of failed edges generated by this coloring. Given a dynamic graph $G = (V, E)$ and a parameter T, for each edge $e \in E$ our dynamic data structure maintains: the round of the edge, i_e, the sequence of colors of the edge, $c_e(1), \ldots, c_e(K)$, and a *color index* $\ell_e \in \{0, \ldots, K\}$. This defines a partition of the edges $S_1, \ldots, S_{T+1}$ where $S_i = \{e \in E \mid i_e = i\}$, and an edge coloring $\tilde{\chi} : E \longrightarrow \mathcal{C} \cup \{\perp\}$ of G where $\tilde{\chi}(e) = c_e(\ell_e)$ and $c_e(0)$ is defined as $\perp$. The edge coloring $\tilde{\chi}$ then defines a set of failed edges $F = \{e \in E \mid \exists f \in N_{i_e}(e)$ such that $\tilde{\chi}(e) = \tilde{\chi}(f)\} \cup \{e \in E \mid \tilde{\chi}(e) = \perp\}$ and a collection of palettes $P_i(u) = [(1+\epsilon)\Delta] \setminus \tilde{\chi}(N_{<i}(u))$, $P_i(u, v) = P_i(u) \cap P_i(v)$. Our data structure maintains all of these objects and supports the following update and query operations.

Updates: The data structure can be updated with the following operations, as described below.

- Insert$(e, i, (c(1), \ldots, c(K)))$: Inserts the edge e into the graph G and assigns the edge e round i, color sequence c, and color index 0.
- Delete(e): Deletes the edge e from the graph G.
- Set-Color-Index(e, ℓ): Sets the color index of the edge e to ℓ.
- Reset-Color(e): Sets the color index of the edge e to the smallest value ℓ such that $c_e(\ell) \in P_i(e)$, and to 0 if no such index exists. Returns the edge e if the color index of e changes, and NULL otherwise.

Queries: The data structure can answer the following types of queries, as described below.

- Node-Palette-Query(u, i, c): The input to this query is a node $u \in V$, an integer $i \in [T]$, and a color $c \in \mathcal{C}$. In response, the data structure outputs YES if $c \in P_i(u)$ and NO otherwise.
- Edge-Palette-Query(e, i, c): The input to this query is an edge $e \in E$, an integer $i \in [T]$, and a color $c \in \mathcal{C}$. In response, the data structure outputs YES if $c \in P_i(e)$ and NO otherwise.
- Failed-Edge-Query(e): The input to this query is an edge $e \in E$. In response, the data structure returns YES if $e \in F$ and NO otherwise.
- Color-Query(e): The input to this query is an edge $e \in E$. In response, the data structure returns the tentative color $\tilde{\chi}(e)$.

We now show how to implement this data structure so that each of these updates and queries run in $O_\epsilon(1)$ expected time. Our data structure also maintains other crucial internal data structures, which we describe below.

Implementation

We create hashmaps $\textsc{Round} : E \longrightarrow [T]$ and $\textsc{Color-Index} : E \longrightarrow [K]$ where $\textsc{Round}(e) = i_e$ and $\textsc{Color-Index}(e) = \ell_e$, allowing is to set and retrieve the rounds and color indices of edges in $O(1)$ (expected) time. The color sequences of edges are implemented as arrays, and we store a hashmap $\textsc{Color-Sequence}$ that maps an edge e to the position of this array, allowing us to retrieve the color $c_e(\ell)$ for an edge $e \in E$ and $\ell \in [K]$ in $O(1)$ time. We implement the set F using a hashmap $\textsc{Failed}$, allowing us to insert, delete, and query the membership of an edge e in $O(1)$ time. Given some edge $e \in E$, we can then compute $\tilde{\chi}(e)$ in $O(1)$ time by retrieving ℓ_e and returning $c_e(\ell_e)$ if $\ell \neq 0$ and $\perp$ otherwise.

Internal Data Structures. Our data structure also maintains the following internal data structures that will be crucial for implementing our algorithm. We maintain hashmaps

$$\phi : V \times [T] \times \mathcal{C} \longrightarrow 2^{N_i(u)} \quad \text{and} \quad \Psi : V \times [T] \times \mathcal{C} \longrightarrow 2^{N_i(u)},$$

such that, for $u \in V$, $i \in [T]$, and $c \in \mathcal{C}$,

$$\phi_{u,i}(c) = \{e \in N_i(u) \mid \tilde{\chi}(e) = c\} \quad \text{and} \quad \Psi_{u,i}(c) = \{e \in N_i(u) \mid c \in c_e\}.$$

We take the convention that $\phi_{u,i}(\perp) = \emptyset$ and $\Psi_{u,i}(\perp) = \emptyset$. We also implement each set $\phi_{u,i}(c)$ as a hashmap, but maintain pointers between the elements of the set forming a doubly linked list. This allows for $O(1)$ time insertions, deletions, and membership queries, while also allowing us to return all the elements in the set in $O(|\phi_{u,i}(c)|)$ time. We implement the sets $\Psi_{u,i}(c)$ in the exact same way. By implementing ϕ as a hashmap, we avoid the preprocessing time of having to initialize each set in $\{\phi_{u,i}(c)\}_{u,i,c}$ upon creating the data structure and only initialize the set $\phi_{u,i}(c)$ when we want to insert an edge into $\phi_{u,i}(c)$, which takes $O(1)$ time. Otherwise, $\phi_{u,i}(c)$ points to $\textsc{null}$, and we know that the set is empty. Similarly, when $\phi_{u,i}(c)$ becomes empty, we can delete the map in $O(1)$ time, so that we only store maps that are non-empty.

Initialization. It follows that, upon initializing our data structure, we only need to create the maps that store the rounds, color sequences, color indices, and failed edges, as well as the 2 maps ϕ and Ψ. This takes $O(1)$ time in total.

Handling Updates

We now show how to maintain our data structure as we perform updates to the graph and the color indices.

Implementing $\textsc{Set-Color-Index}$**.** We first note that changing the color index ℓ_e of an edge e will *not* change the set $\Psi_{u,i}(c)$ or whether or not an edge $f \neq e$ is contained in the set $\phi_{u,i}(c)$ for any $u \in V$, $i \in [T]$, and $c \in \mathcal{C}$. Hence, in order to update the $\phi_{u,i}$ and $\Psi_{u,i}$ maps after a color index update for edge e, we only need to insert and remove e from the appropriate sets $\phi_{u,i}(c)$. On

the other hand, some $O(1)$ many edges neighboring e might have to be added or removed from F after a color index update for e. Algorithm 24 shows how we implement the Set-Color-Index procedure in order to appropriately update all the data structures used by our algorithm.

Algorithm 24: Set-Color-Index(e, ℓ)

▷ Let u and v be the endpoints of e and $i = i_e$
$\ell_e^{\text{PREV}} \leftarrow \ell_e$ and $c^{\text{PREV}} \leftarrow \tilde{\chi}(e)$
$\ell_e \leftarrow \ell$ and $c \leftarrow \tilde{\chi}(e)$
▷ Update the ϕ maps
if $c^{PREV} \neq \perp$ **then**
    $\phi_{u,i}(c^{\text{PREV}}) \leftarrow \phi_{u,i}(c^{\text{PREV}}) \setminus \{e\}$
    $\phi_{v,i}(c^{\text{PREV}}) \leftarrow \phi_{v,i}(c^{\text{PREV}}) \setminus \{e\}$
if $c \neq \perp$ **then**
    $\phi_{u,i}(c) \leftarrow \phi_{u,i}(c) \cup \{e\}$
    $\phi_{v,i}(c) \leftarrow \phi_{v,i}(c) \cup \{e\}$
▷ Update the set of failed edges F
$X \leftarrow \{e\}$
if $|\phi_{u,i}(c)| = 2$ **then**
    $X \leftarrow X \cup \phi_{u,i}(c)$
if $|\phi_{v,i}(c)| = 2$ **then**
    $X \leftarrow X \cup \phi_{v,i}(c)$
if $|\phi_{u,i}(c^{PREV})| = 1$ **then**
    $X \leftarrow X \cup \phi_{u,i}(c^{\text{PREV}})$
if $|\phi_{v,i}(c^{PREV})| = 1$ **then**
    $X \leftarrow X \cup \phi_{v,i}(c^{\text{PREV}})$
for $f \in X$ **do**
    if $|\phi_{u,i}(\tilde{\chi}(f))| > 1$ or $|\phi_{v,i}(\tilde{\chi}(f))| > 1$ **then**
        $F \leftarrow F \cup \{f\}$
    else
        $F \leftarrow F \setminus \{f\}$

Lemma 10.3.7. *The implementation of Set-Color-Index given by Algorithm 24 correctly updates the data structure and runs in time $O(1)$.*

Proof. Before the update, edge e is contained in both $\phi_{u,i}(c^{\text{PREV}})$ and $\phi_{v,i}(c^{\text{PREV}})$, and no other such sets. After the update, since $\tilde{\chi}(e)$ changes to c, we remove e from these sets and add it to $\phi_{u,i}(c)$ and $\phi_{v,i}(c)$, taking $O(1)$ time. After performing these operations, the ϕ maps are now correctly updated for the new tentative coloring. In order to see that our algorithm correctly updates the set F, note that the only edges that might need to be removed from F are those incident on u and v at round i that have tentative color c^{PREV}. However, if there is more than 1 edge incident on u (resp. v) at round i with tentative color c^{PREV} after updating the map ϕ, then no edge incident

on u (resp. v) will be removed from F as all these edges will fail. Similarly, the only edges that might need to be added to F are those incident on u and v at round i that have tentative color c. However, if there are more than 2 edges incident on u (resp. v) at round i with tentative color c after updating the ϕ maps, then no edge incident on u (resp. v) will be added to F since these edges will have failed before the update. It follows that a total of $O(1)$ many edges might need to be added or removed from F and that we can identify them in $O(1)$ time. We can scan through all of these edges and for each one determine in $O(1)$ time whether it should or shouldn't be in F by checking if $|\phi_{u,i}(\tilde{\chi}(f))| > 1$ or $|\phi_{v,i}(\tilde{\chi}(f))| > 1$ which is true if and only f fails. □

Implementing Insert **and** Delete**.** We first note that inserting or deleting an edge $e = (u, v)$ that has tentative color $\perp$ will not cause any change in the map ϕ. Furthermore, it will not cause any edge $f \neq e$ to added or removed from F. Hence, we can insert or delete such edges and update the data structures easily. In order to delete an edge e that has $\tilde{\chi}(e) \neq \perp$, we can first call Set-Color-Index$(e, 0)$ and then assume we are deleting an edge with tentative color $\perp$. Algorithms 25 and 26 show how we can implement this. These algorithms clearly update all the data structures correctly and can be implemented to run in $O(K)$ time.

Algorithm 25: Insert$(e, i, (c(1), \ldots, c(K)))$

$i_e \leftarrow i$
$c_e \leftarrow (c(1), \ldots, c(K))$
$\ell_e \leftarrow 0$
$F \leftarrow F \cup \{e\}$
for $\ell' = 1 \ldots K$ **do**
  $\Psi_{u,i}(c_e(\ell')) \leftarrow \Psi_{u,i}(c_e(\ell')) \cup \{e\}$
  $\Psi_{v,i}(c_e(\ell')) \leftarrow \Psi_{v,i}(c_e(\ell')) \cup \{e\}$

Algorithm 26: Delete(e)

Set-Color-Index$(e, 0)$
$i_e \leftarrow$ NULL
$c_e \leftarrow$ NULL
$\ell_e \leftarrow$ NULL
$F \leftarrow F \setminus \{e\}$
for $\ell' = 1 \ldots K$ **do**
  $\Psi_{u,i}(c_e(\ell')) \leftarrow \Psi_{u,i}(c_e(\ell')) \setminus \{e\}$
  $\Psi_{v,i}(c_e(\ell')) \leftarrow \Psi_{v,i}(c_e(\ell')) \setminus \{e\}$

Implementing Reset-color**.** We implement the Reset-color update by scanning through the list of colors c_e and finding the smallest ℓ such that $c_e(\ell) \in P_{i_e}(e)$ by making calls to Edge-Palette-Query. Once we identify the smallest such ℓ, we call Set-Color-Index(e, ℓ) and return e if ℓ_e changes. If we cannot find such an index ℓ, we call Set-Color-Index$(e, 0)$ and return e if ℓ_e changes. As we will see in Section 11, each edge palette query can be implemented to run in time

$O(T)$. Since we make at most K such queries, one call to SET-COLOR-INDEX that takes $O(1)$ time, and the rest of the operations can be implemented in $O(1)$ time, it follows that RESET-COLOR can be implemented to run in $O(KT)$ time. Algorithm 27 shows how we can implement this algorithm.

Algorithm 27: RESET-COLOR(e)

```
ℓ' ← ℓ_e
for ℓ = 1, ..., K do
    if EDGE-PALETTE-QUERY(e, i_e, c_e(ℓ)) = YES then
        SET-COLOR-INDEX(e, ℓ)
        if ℓ' ≠ ℓ then
            return e
        return NULL
SET-COLOR-INDEX(e, 0)
if ℓ' ≠ 0 then
    return e
return NULL
```

Answering the Queries

Since we maintain the set F as a hashmap, given some edge e, we can answer FAILED-EDGE-QUERY on edge e in $O(1)$ time by directly checking whether the edge e is contained in F. Given some $u \in V$, $i \in [T]$, and $c \in \mathcal{C}$, we can check whether $c \in P_i(u) = [(1+\epsilon)\Delta] \setminus \tilde{\chi}(N_{<i}(u))$ by checking whether $\phi_{u,i'}(c) = \emptyset$ for all $i' < i$, which can be done in $O(i) \leq O(T)$ time. Hence, we can answer the query NODE-PALETTE-QUERY in $O(T)$ time. Given some $e = (u, v) \in E$, $i \in [T]$, and $c \in \mathcal{C}$, we can check whether $c \in P_i(e)$ by making 2 calls to NODE-PALETTE-QUERY and checking whether NODE-PALETTE-QUERY $c \in P_i(u)$ and $c \in P_i(v)$. Hence, we can answer the query EDGE-PALETTE-QUERY in $O(T)$ time. The fact that COLOR-QUERY can be implemented in $O(1)$ time follows immediately from the implementation.

10.3.3 Proof of Lemma 10.3.2

Let $G = (V, E)$ be a dynamic graph that undergoes updates via a sequence of edge insertions and deletions, and let Δ be an upper bound on the maximum degree of G at any point in time. We now show how our data structure can be used to explicitly maintain the tentative coloring $\tilde{\chi}^{(t)}$ and set of failed edges $F^{(t)}$ defined by running Algorithm 22 on graph $G^{(t)}$ with parameters Δ and ϵ. Let i_e and c_e denote the round and color sequence sampled in advance for potential edge $e \in \binom{V}{2}$.

We first remark that, if we have that our data structure currently stores the graph $G^{(t-1)}$, each edge e receives the color sequence c_e and round i_e, and that the color index ℓ_e of each edge e equals the color index $\ell_e^{(t-1)}$ defined by Algorithm 22, then clearly the tentative coloring $\tilde{\chi}$ and the set of failed edges F maintained by our data structure equal $\tilde{\chi}^{(t-1)}$ and $F^{(t-1)}$. Hence, in order to be able to maintain $\tilde{\chi}^{(t-1)}$ and $F^{(t-1)}$, given the edge $e^\star$ that is either inserted or deleted during the t^{th}

update, we need to appropriately update the graph maintained by our data structure (by inserting or deleting $e^\star$) and then update the color indices so that $\ell_e = \ell_e^{(t)}$ for all edges e. Given that our data structure is in the state described above, we now show how to do this efficiently.

Updating the Data Structure. When an edge $e^\star$ is inserted into the graph G, we run Algorithm 28 on input $e^\star$, which passes $e^\star$ to the data structure along with its round $i_{e^\star}$ and color sequence $c_{e^\star}$. When an edge $e^\star$ is deleted from the graph G, we run Algorithm 29 on input $e^\star$, which removes $e^\star$ from the data structure. In both cases, the algorithm then determines if $e^\star$ is dirty and then passes it to Algorithm 30 if this is the case. Algorithm 30 takes the set $A_{i_{e^\star}}^{(t)}$ and round $i_{e^\star}$ as inputs, where $i_{e^\star}$ is the first round containing dirty edges, and propagates the changes in the tentative coloring through the rounds. After the update is complete, we have that $\ell_e = \ell_e^{(t)}$ for all edges e in the graph.

Algorithm 28: INSERTION-UPDATE($e^\star$)

$\tilde{\chi}^{\text{PREV}}(e^\star) \leftarrow \perp$
INSERT($e^\star, i_{e^\star}, c_{e^\star}$)
$X \leftarrow \{\text{RESET-COLOR}(e^\star)\}$
PROPAGATE-CHANGES($X, i_{e^\star}$)

Algorithm 29: DELETION-UPDATE($e^\star$)

$\tilde{\chi}^{\text{PREV}}(e^\star) \leftarrow \tilde{\chi}(e^\star)$
DELETE($e^\star$)
$X \leftarrow \emptyset$
if $\tilde{\chi}^{PREV}(e^\star) \neq \perp$ **then**
  $X \leftarrow X \cup \{e^\star\}$
PROPAGATE-CHANGES($X, i_{e^\star}$)

Algorithm 30: PROPAGATE-CHANGES($X_{i'}, i'$)

for $i = i' + 1, \ldots, T$ **do**
  $X_i' \leftarrow \bigcup_{e \in X_{i-1}} \Psi_{e,i}(\tilde{\chi}(e)) \cup \Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e))$
  $\tilde{\chi}^{\text{PREV}}(e) \leftarrow \tilde{\chi}(e)$ for all $e \in X_i'$
  $X_i \leftarrow X_{i-1}$
  for $e \in X_i'$ **do**
    $X_i \leftarrow X_i \cup \text{RESET-COLOR}(e)$

Here we let $\Psi_{e,i}(c)$ denote the set $\Psi_{u,i}(c) \cup \Psi_{v,i}(c)$, where $e = (u, v)$.

Correctness

Suppose that we call INSERTION-UPDATE($e^\star$) (resp. DELETION-UPDATE($e^\star$)) to update the data structure after the insertion (resp. deletion) of the edge $e^\star$ at time t. The following lemmas describe the behavior of our algorithm. We denote by ℓ_e and $P_i(u)$ the color indices and palettes maintained by our algorithm and by $\ell_e^{(t)}$ and $P_i^{(t)}(u)$ the color indices and palettes defined by Algorithm 22

on input $G^{(t)}$. Our objective is to show that $\ell_e = \ell_e^{(t)}$ for all edges e *after* the update is complete. Recall that we assume $\ell_e = \ell_e^{(t-1)}$ for all edges e *before* we the perform the update.

Lemma 10.3.8. *Given any $X \subseteq S_{<i}^{(t)}$, we have that*

$$\Gamma_i^{(t)}(X) \cup \Lambda_i^{(t)}(X) \subseteq \bigcup_{e \in X} \Psi_{e,i}(\tilde{\chi}^{(t)}(e)) \cup \Psi_{e,i}(\tilde{\chi}^{(t-1)}(e)).$$

Proof. Let $e \in \Gamma_i^{(t)}(X)$. Then there exists some $f \in X$ such that $\tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t-1)}(e)$, so $\tilde{\chi}^{(t)}(f) \in c_e$ and hence $e \in \Psi_{f,i}(\tilde{\chi}^{(t)}(f))$. Now let $e \in \Lambda_i^{(t)}(X)$. Then there exists some $f \in X$ such that $\tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e)$, so $\tilde{\chi}^{(t-1)}(f) \in c_e$ and hence $e \in \Psi_{f,i}(\tilde{\chi}^{(t)}(f))$. □

Lemma 10.3.9. *For all $i \in [T]$, if $\ell_e = \ell_e^{(t)}$ for all $e \in S_{<i}^{(t)}$, then running* Recolor-Edge *on any $e \in S_i^{(t)}$ will set $\ell_e = \ell_e^{(t)}$.*

Proof. Since $\ell_e = \ell_e^{(t)}$ for all $e \in S_{<i}^{(t)}$, it follows that $\tilde{\chi}(e) = \tilde{\chi}^{(t)}(e)$ for all $e \in S_{<i}^{(t)}$, so $P_i(u) = P_i^{(t)}(u)$ for all $u \in V$. Given any edge $e \in S_i^{(t)}$, we then have that $P_i(e) = P_i^{(t)}(e)$, so when we run Recolor-Edge(e) we set the color index ℓ_e to the smallest ℓ such that $c_e(\ell) \in P_i^{(t)}(e)$ (or 0 if no such ℓ exists), which is the exact definition of $\ell_e^{(t)}$. □

Lemma 10.3.10. *For all $i \geq i_e$, we have that $X_i = A_{\leq i}^{(t)}$. Furthermore, after our update procedure terminates, we have that $\ell_e = \ell_e^{(t)}$ for all $e \in E^{(t)}$.*

Proof. We prove this by induction. For all $i \geq i_{e^\star} + 1$, we show that the following are true at the start of the ith iteration of the **for** loop in Algorithm 30 (where we start from iteration $i_{e^\star} + 1$):

1. $X_{i-1} = A_{<i}^{(t)}$, and
2. $\ell_e = \ell_e^{(t)}$ for all $e \in S_{<i}^{(t)}$.

Base Case. We begin by showing that these conditions all hold for $i = i_{e^\star} + 1$. Let $i^\star = i_{e^\star}$. We first show that $X_{i^\star} = A_{\leq i^\star}^{(t)}$. In the event that the t^{th} update is an insertion, we have that $X_{i^\star}$ is empty if $\tilde{\chi}^{(t)}(e^\star) = \perp$ (note that Reset-Color$(e^\star)$ returns $e^\star$ if and only if this is not the case), in which case the set $A_{\leq i^\star}^{(t)}$ is also empty, and $X_{i^\star} = \{e^\star\}$ if $\tilde{\chi}^{(t)}(e^\star) \neq \perp$, in which case $A_{\leq i^\star}^{(t)} = \{e^\star\}$. Similarly, in the event that the t^{th} update is an deletion, we have that $X_{i^\star}$ is empty if $\tilde{\chi}^{(t-1)}(e^\star) = \perp$, in which case $A_{\leq i^\star}^{(t)}$ is also empty, and $X_{i^\star} = \{e^\star\}$ if $\tilde{\chi}^{(t-1)}(e) \neq \perp$, in which case $A_{\leq i_e}^{(t)} = \{e^\star\}$. To see that $\ell_e = \ell_e^{(t)}$ for all $e \in S_{\leq i^\star}^{(t)}$ at the start of iteration $i^\star + 1$, note that $\ell_e = \ell_e^{(t)}$ for all $e \in S_{<i^\star}^{(t)}$ when we call Reset-Color$(e^\star)$ (since there are no dirty edges in the first $i^\star - 1$ rounds). Hence, by Lemma 10.3.9, we have that $\ell_{e^\star} = \ell_{e^\star}^{(t)}$ at the start of iteration $i^\star + 1$. As this is the only edge in round $i^\star$ that can change its tentative color, the claim follows.

Inductive Step. Suppose that the inductive hypothesis holds for $i_e + 1 \leq i \leq T$. We now show that it also holds for $i + 1$. By Lemma 10.3.8 and Corollary 10.2.9, we can see that

$$X_i' \supseteq \Gamma_i^{(t)}(X_{i-1}) \cup \Lambda_i^{(t)}(X_{i-1}) = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)}) = A_i^{(t)}.$$

Since the algorithm scans through all of the edges in X_i' and calls RECOLOR-EDGE on each of them, and we have that $\ell_e = \ell_e^{(t)}$ for all $e \in S_{<i}^{(t)}$, it follows by Lemma 10.3.9 that we have $\ell_e = \ell_e^{(t)}$ for all $e \in S_{\leq i}^{(t)}$ at the end of the iteration (note that X_i' contains all of the edges at round i that change their color indices during this update). The algorithm places all of the edges $e \in X_i'$ that change their color index ℓ_e after calling RECOLOR-EDGE(e) into X_i along with the edges in $A_{<i}^{(t)}$. Since $X_i' \subseteq A_i^{(t)}$, it follows that $X_i = A_{\leq i}^{(t)}$ at the end of the iteration. □

Update Time Analysis

Lemma 10.3.11. *For all $i > i_{e^\star}$, $\mathbb{E}[|X_i'|] \leq (32/\epsilon)\log(1/\epsilon) \cdot |X_{i-1}|$.*

Proof. For all $i > i_e$,

$$X_i' = \bigcup_{e \in X_{i-1}} \Psi_{e,i}(\tilde{\chi}(e)) \cup \Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e)),$$

which implies that

$$|X_i'| \leq \sum_{e \in X_{i-1}} |\Psi_{e,i}(\tilde{\chi}(e))| + |\Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e))|.$$

Let $u \in V$, $i \in [T]$, and $c \in [(1+\epsilon)\Delta]$. Then for all $e \in N_i(u)$ we have that

$$\Pr[e \in \Psi_{u,i}(c)] = \Pr[c \in c_e] \leq \frac{K}{(1+\epsilon)\Delta}$$

since c_e is a sequence of K colors sampled independently and uniformly at random from $[(1+\epsilon)\Delta]$. By linearity of expectation we get that $\mathbb{E}[|\Psi_{u,i}(c)|] \leq |N_i(u)| \cdot K/((1+\epsilon)\Delta)$. Taking expectations on both sides and noting that $\mathbb{E}[|N_i(u)|] \leq \epsilon(1-\epsilon)^{i-1}\Delta$ (see Lemma 10.1.9), we get that

$$\mathbb{E}[|\Psi_{u,i}(c)|] \leq \mathbb{E}[|N_i(u)|] \cdot \frac{K}{(1+\epsilon)\Delta} \leq \epsilon K = \frac{8}{\epsilon}\log\frac{1}{\epsilon}.$$

It follows that

$$\mathbb{E}[|X_i'|] \leq \sum_{e \in X_{i-1}} \mathbb{E}[|\Psi_{e,i}(\tilde{\chi}(e))|] + \mathbb{E}[|\Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e))|] \leq |X_{i-1}| \cdot \frac{32}{\epsilon}\log\frac{1}{\epsilon}.$$ □

Lemma 10.3.12. *Algorithms 28 and 29 run in $O(\log^4(1/\epsilon)/\epsilon^5 \cdot \mathbb{E}[|A^{(t)}|])$ expected time.*

Proof. Algorithms 28 and 29 run in $O(KT)$ time excluding the call to PROPAGATE-CHANGES$(X, i_{e^\star})$, since the calls to INSERT and DELETE take $O(K)$ time, the call to RESET-COLOR$(e^\star)$ takes $O(KT)$ time, and the rest of the operations run in $O(1)$ time. The i^{th} iteration of Algorithm 30 (where we start from iteration $i_{e^\star}+1$) takes time $O(|X_{i-1}| + KT \cdot |X_i'|)$. Since we can return the sets $\Psi_{u,i}(c)$ in time proportional to their size, Line 2 runs in $O(|X_i'| + |X_{i-1}|)$ time. Lines 3-4 also run in time $O(|X_i'| + |X_{i-1}|)$. Lines 5-6 take $O(KT \cdot |X_i'|)$. It follows that the running time of these algorithms

is

$$O(KT) + \sum_{i=i^\star+1}^{T} O(|X_{i-1}| + KT \cdot |X'_i|).$$

Taking expectations, applying Lemma 10.3.11, and then taking expectations again, it follows that the expected running time of these algorithms is

$$O(KT) + \sum_{i=i^\star+1}^{T} O\left(\frac{1}{\epsilon^4}\log^3\frac{1}{\epsilon} \cdot \mathbb{E}[|X_{i-1}|]\right) = O\left(\frac{1}{\epsilon^4}\log^3\frac{1}{\epsilon}\right) \cdot \sum_{i=i^\star+1}^{T} \mathbb{E}[|X_{i-1}|].$$

Applying Lemma 10.3.10, we can upper bound this by

$$O\left(\frac{1}{\epsilon^4}\log^3\frac{1}{\epsilon}\right) \cdot T \cdot \mathbb{E}[|A^{(t)}_{\leq T}|] \leq O\left(\frac{1}{\epsilon^5}\log^4\frac{1}{\epsilon} \cdot \mathbb{E}[|A^{(t)}|]\right). \qquad \square$$

10.3.4 Proof of Lemma 10.3.3

Let $0 < \delta \leq 1$ be a constant. Then we have the following lemma.

Lemma 10.3.13. *There exists a dynamic data structure that, given a dynamic graph $G = (V, E)$ that undergoes edge insertions and deletions, can explicitly maintain a $(2+\delta)\Delta(G)$-edge coloring of G, has an expected update time of $O(1/\delta)$, and only changes the colors of $O(1)$ many edges per update.*

Proof. Our data structure maintains the graph G in a manner that allows edges to be inserted and deleted in $O(1)$ time (see Section 10.3.2), and the coloring $\chi : E \longrightarrow [(2+\delta)\Delta(G)]$ using a hashmap, allowing us to retrieve and set a color $\chi(e)$ in $O(1)$ time for any $e \in E$. Similarly, it maintains a hashmap $\psi : V \times [(2+\delta)\Delta(G)] \longrightarrow E$ that maps nodes u and a color c to the edge e incident on u with $\chi(e) = c$, if such an edge exists. This algorithm maintains the invariant that each edge $e = (u, v)$ receives a color from $[(2+\delta)\max\{\deg(u), \deg(v)\}]$.

Inserting an Edge. When we insert an edge $e = (u, v)$ into the graph G, we sample a color c independently and u.a.r. from $[(2+\delta)\max\{\deg(u), \deg(v)\}]$. Let $d^\star := \max\{\deg(u), \deg(v)\}$. The probability that this color is available at e, i.e. is contained in the palette $P(e) = P(u) \cap P(v)$ where $P(u) = [(2+\delta)d^\star] \setminus \chi(N(u))$, is at least

$$\Pr[c \in P(e)] = \frac{|P(e)|}{(2+\delta)d^\star} = \frac{|P(u)| + |P(v)| - |P(u) \cup P(v)|}{(2+\delta)d^\star}$$

$$\geq \frac{(2+\delta)d^\star - \deg(u) + (2+\delta)d^\star - \deg(v) - (2+\delta)d^\star}{(2+\delta)d^\star} \geq \frac{\delta}{(2+\delta)} \geq \frac{\delta}{3}.$$

Using the hashmaps ψ_u and ψ_v we can check whether $c \in P(e)$ in $O(1)$ time. If $c \notin P(e)$, we sample another color in the same way, and repeat until we find a color $c \in P(e)$. Since these events that the sampled colors are in $P(e)$ are independent and all succeed with probability at least $\delta/3$, it follows that the expected number of colors that we have to sample before finding one in $P(e)$ is $3/\delta$. Hence,

it takes us $O(1/\delta)$ time to find a color $c \in P(e)$ in expectation. Once we have found such a color, we set $\chi(e) \leftarrow c$, $\psi_u(c) \leftarrow e$, and $\psi_v(c) \leftarrow e$. Since the degrees of nodes cannot decrease during an edge insertion, the invariant is still satisfied.

Deleting an Edge. When we delete an edge $e = (u, v)$ from the graph G, we first set $\psi_u(\chi(e)) \leftarrow$ NULL and $\psi_v(\chi(e)) \leftarrow$ NULL, and then set $\chi(e) \leftarrow$ NULL. However, the degrees of u and v decrease by one, so there might be edges that no longer satisfy the invariant. In particular, any edge f incident on u that no longer satisfies the invariant will have $\chi(f) \in [(2+\delta)(\deg(u)+1)] \setminus [(2+\delta)\deg(u)]$. Since there are at most 3 such colors, we can use the hashmap ψ_u to identify the edges incident on u that receive one of these colors in $O(1)$ time. We then uncolor these edges, while ensuring to appropriately update the hashmaps, and recolor them using the same strategy as outlined above for an edge insertion. We do the same thing at node v. In total, we recolor at most 6 edges in G to ensure that we still satisfy the invariant, taking $O(1/\delta)$ time in expectation. □

10.4 Implementing our Static Algorithm

We now show how to implement our static algorithm directly in order to obtain better performance than what follows immediately from our dynamic algorithm. We first show how to get linear time in expectation, and then expand on this to get linear time with high probability. The main result in this section is Theorem 10.4.5, which is restated below.

Theorem 10.4.1. *There exists an algorithm that, given a graph G with maximum degree Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, returns a $(1+61\epsilon)\Delta$-edge coloring of G in $O(m \log(1/\epsilon)/\epsilon^2)$ time with probability at least $1 - O(1/n^6)$.*

10.4.1 Linear Time in Expectation

It follows immediately from Corollary 10.3.5 that our static algorithm can be implemented to run in $O(m \log^4(1/\epsilon)/\epsilon^9)$ expected time. We now show how to implement our static algorithm more directly using the data structure from Section 10.3 in order to get the following theorem.

Theorem 10.4.2. *There exists an algorithm that, given a graph G with maximum degree Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, returns a $(1+61\epsilon)\Delta$-edge coloring of G in $O(m \log(1/\epsilon)/\epsilon^2)$ expected time.*

Proof. We begin by using our data structure as a black box in order to show how our static algorithm can be implemented to run in $O(KTm)$ expected time. We then show how a white-box application can improve this to $O(Km)$ expected time.

Given the graph $G = (V, E)$, we begin by splitting the edges E into $\mathcal{E}_1, \ldots, \mathcal{E}_\eta$ as defined in Algorithm 21. We do this by creating lists to store each of the $\mathcal{E}_j$ and scan through all of the edges in $e \in E$, assigning each to one of the $\mathcal{E}_j$ independently and u.a.r., taking $O(m)$ time. If we can now show how to implement Algorithm 22 so that it runs in expected time $g(m')$ on an input graph with

m' edges, it follows that we can compute the failed edges and the tentative coloring in expected time $\sum_{j\in[\eta]} g(|\mathcal{E}_j|)$. By then noticing that Lemma 10.3.13 immediately implies an expected linear time greedy algorithm, we get that the static algorithm runs in expected time $O(m)+\sum_{j\in[\eta]} g(|\mathcal{E}_j|)$. If the function g is linear and $g(x) = \Omega(x)$ (as it will be in the following cases), this then gives an expected running time of $O(g(m))$. This algorithm produces a $(1 + 61\epsilon)\Delta$-edge coloring with probability at least $1 - O(1/n^6)$. If it uses more than $(1 + 61\epsilon)\Delta$ many colors, we can afford to keep rerunning it using fresh randomness until we use at most $(1 + 61\epsilon)\Delta$ many colors, without increasing the asymptotic expected running time. (see Section 10.3 for a precise description of how to implement the resampling efficiently).

Black-Box Application of Our Data Structure. Now we show how to implement Algorithm 22 so that it runs in $O(KTm)$ expected time given a graph $G = (V, E)$ with m edges. We begin by splitting the set E into $S_1, \ldots, S_{T+1}$ by sampling a round i_e for each edge $e \in E$ and then placing e into a list containing the edges that are colored at round i_e, again taking $O(m)$ time in total. We then proceed to initialize our data structure, which takes $O(1)$ time. For $i = 1 \ldots T$, we then insert all of the edges in S_i into the data structure (in any order) so that they are assigned a color index of 0 and hence left uncolored, and then call Reset-Color on each edge in S_i (again, in any order). By Lemma 10.3.9, we have (by induction) that this correctly computes the color indices of all the edges in S_i as defined by the color sequences and rounds that we generate for the edges. Hence, we also compute the correct tentative colors and set of failed edges. Since Reset-Color runs in time $O(KT)$, and it takes $O(K)$ time to insert an edge into the data structure, it follows that the algorithm runs in time $O(KTm)$ since must do this for each edge exactly once.

White-Box Application of Our Data Structure. In order to improve this to $O(Km)$, we notice that in this setting we can implement the query Edge-Palette-Query to run in in $O(1)$ expected time, improving the runtime of Reset-Color to $O(K)$, which gives the result. We do this by noticing that we can maintain a hashmap $\phi' : V \times \mathcal{C} \longrightarrow 2^{N(u)}$ where $\phi'_u(c) = \{e \in N(u) \mid \tilde{\chi}(e) = c\}$ and each $\phi'_u(c)$ is implemented in the same way as the $\phi_{u,i}(c)$. Since $\phi'_u(c) = \bigcup_{i=1}^{T} \phi_{u,i}(c)$, we can maintain the map ϕ' by appropriately updating the set $\phi'_u(c)$ every time we update one the sets $\phi_{u,i}(c)$, incurring only $O(1)$ overhead. Since this $O(1)$ overhead does not change the asymptotic behavior of our data structure, this does not change the asymptotic running time of any of the updates or queries. When our algorithm now makes a call to Edge-Palette-Query after calling Reset-Color on an edge $e = (u, v)$ appearing in round i, we note that no edges in $S_{>i}$ have been inserted into the graph yet. Hence, we have that a color c is contained in $P_i(e)$ iff $|\phi'_u(c)| - |\phi_{u,i}(c)| = 0$ and $|\phi'_v(c)| - |\phi_{v,i}(c)| = 0$, which we can check in $O(1)$ time. □

10.4.2 Linear Time with High Probability

In order to obtain an algorithm that runs in $O_\epsilon(m)$ time with high probability, we need to obtain concentration. Our first obstacle is our use of hashmaps. We need to argue that we can implement these hashmaps so that not only can we handle insert, delete, and query operations in $O(1)$ expected

time, but also so that these operations all take $O(1)$ time with high probability. For this, we use the following lemma which follows from [DadH90].

Lemma 10.4.3 (Theorem 5.5, [DadH90])**.** *There exists a dynamic dictionary that, given a parameter k, can handle k insertion, deletion, and query operations, uses $O(k)$ space, and takes $O(1)$ worst-case time per operation with probability at least $1 - O(1/k^7)$.*

The second obstacle comes from our $O(\Delta)$ greedy algorithm, where we can make an unbounded number of queries to a hashmap. By applying the following concentration inequality for sums of geometric random variables given in [Bro11], we show that we only perform $O(m)$ many queries to the hashmap with high probability.

Lemma 10.4.4. *Let $X_1, \ldots, X_n$ be n independent geometric random variables with success probability p, and let $X = \sum_i X_i$. Then, for $0 < \epsilon \leq 1$, we have that*

$$\Pr[X > (1+\epsilon)\mathbb{E}[X]] \leq \exp\left(-\frac{\epsilon^2}{4}n\right).$$

By combining these tools, we can analyze our static algorithm more carefully, and get the following result.

Theorem 10.4.5. *There exists an algorithm that, given a graph G with maximum degree Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, returns a $(1+61\epsilon)\Delta$-edge coloring of G in $O(m \log(1/\epsilon)/\epsilon^2)$ time with probability at least $1 - O(1/n^6)$.*

Proof. We begin by showing how we can use the dynamic dictionary from Lemma 10.4.3 to implement the hashmaps in Algorithm 22 so that each insertion, deletion, and query made by the algorithm which we described in the preceding section takes $O(1)$ time with probability at least $1 - O(1/m^7)$, where m is the number of edges in the input graph $G = (V, E)$. This will immediately imply that we can implement Algorithm 22 to run in $O(Km)$ time with probability at least $1 - O(1/m^7)$. We first note that our data structure uses the maps ϕ, ϕ', Round, Color-Index, Color-Sequence, and Failed. Our implementation of Algorithm 22 does not need to use the map Ψ, so we can ignore all operations performed on this map in this context. As for the sets $\phi_{u,i}(c)$, note that they can only increase in size as we run the algorithm. Since the algorithm never accesses the elements in $\phi_{u,i}(c)$ after its size exceeds 2, we do not need to store the set once it becomes large enough, and instead we just store its size after this point. Hence, we do not need to implement them as hashmaps and can implement all operations on the set $\phi_{u,i}(c)$ in $O(1)$ worst-case time. The same applies to the sets $\phi'_u(c)$. Since we never need to access the elements in $\phi'_u(c)$, it is sufficient to simply keep track of its size, which can easily be achieved with a counter that can be updated in $O(1)$ time.

For each edge $e \in E$, we make one call to Insert(e) and one call to Reset-Color(e). It follows that, throughout the entire run of Algorithm 22, we perform $O(Km)$ many operations on the map ϕ' and $O(m)$ many operations on each of the other maps. Hence, we can implement

each of these maps using the dynamic dictionary in Lemma 10.4.3, initializing each one with a parameter of size $O(Km)$. It follows from a union bound that all the operations performed on all the hashmaps throughout the run of Algorithm 22 takes $O(1)$ time with probability at least $1-O(1/(Km)^7) \geq 1-O(1/n^7)$. By the arguments in the preceding section, it follows that the total time taken to subsample the graph and handle each call to Algorithm 22 for each of the subsampled graphs is $O(Km)$ with probability at least $1 - O(\eta/n^7) \geq 1 - O(1/n^6)$.

In order to implement our greedy algorithm so that it runs in $O(m)$ time with high probability, we again implement the hashmap ψ used by the algorithm using the dynamic dictionary in Lemma 10.4.3, passing a sufficiently large parameter of order $O(m)$. However, it is not immediately clear that the algorithm performs at most $O(m)$ operations on the map ψ. We can observe that the insertion of the edge e into the greedy algorithm leads to $O(X_e)$ many operations on the map ψ, where X_e is a random variable denoting the number of colors sampled by the algorithm while trying to sample a color from the palette of edge e. Clearly, the value of X_e is $O(1)$ in expectation. It follows that, in expectation, we perform $\mathbb{E}[O(\sum_e X_e)] \leq O(m)$ many operations on the map ψ. In order to establish concentration, we can observe that $\{X_e\}_e$ is a collection of independent geometric random variables. Thus, we can apply the concentration inequality from Lemma 10.4.4 to get that our algorithm performs at most $O(m)$ many operations on ψ with probability at least $1 - e^{-\Theta(m)}$. Conditioned on this event, all of the operations performed on our hashmap run in $O(1)$ worst-case time with probability at least $1 - O(1/m^7)$. Hence, the greedy algorithm runs in $O(m)$ time with probability at least $1 - O(1/m^7)$.

Putting everything together, we get that our static algorithm runs in $O(Km)$ time with probability at least $1-O(1/n^6)$. Since the algorithm returns a $(1+61\epsilon)\Delta$-edge coloring with probability at least $1-O(1/n^6)$, the lemma follows. □

10.5 Concentration of Measure

In this section, we state the probabilistic tools that we use to establish concentration of measure throughout this chapter. This section is essentially a subset of Appendix E in [BGW21].

10.5.1 Concentration Bounds

We now introduce some standard concentration bounds for independent random variables. The proofs of all of these bounds can be found in [DP09].

Proposition 10.5.1 (Chernoff Bounds)**.** *Let X be the sum of n mutually independent indicator random variables $X_1, \ldots, X_n$. Then, for any $\mu_L \leq \mathbb{E}[X] \leq \mu_H$, for all $\epsilon > 0$, we have that*

$$\Pr[X > (1+\epsilon)\mu_H] \leq \exp\left(-\frac{\epsilon^2}{3}\mu_H\right),$$

$$\Pr[X < (1-\epsilon)\mu_L] \leq \exp\left(-\frac{\epsilon^2}{2}\mu_L\right).$$

Proposition 10.5.2 (Hoeffding Bounds)**.** *Let X be the sum of n mutually independent indicator random variables $X_1, \ldots, X_n$. Then, for all $t > 0$, we have that*

$$\Pr[X > \mathbb{E}[X] + t] \leq e^{-2t^2/n},$$

$$\Pr[X < \mathbb{E}[X] - t] \leq e^{-2t^2/n}.$$

Definition 10.5.3 (Lipschitz Functions)**.** *Consider n sets $A_1, \ldots, A_n$ and a real valued function $f : A_1, \ldots, A_n \longrightarrow \mathbb{R}$. The function f satisfies the Lipschitz property with constants $d_1, \ldots, d_n$ if and only if $|f(x) - f(y)| \leq d_i$ whenever x and y differ only in the i*th *coordinate, for all $i \in [n]$.*

Proposition 10.5.4 (Method of Bounded Differences)**.** *If f satisfies the Lipschitz property with constants $d_1, \ldots, d_n$, and $X_1, \ldots, X_n$ are independent random variables, then, for all $t > 0$, we have that*

$$\Pr[f < \mathbb{E}[f] + t] \leq \exp\left(\frac{2t^2}{d}\right),$$

$$\Pr[f > \mathbb{E}[f] - t] \leq \exp\left(\frac{2t^2}{d}\right),$$

where $d = \sum_i d_i^2$.

10.5.2 Negatively Associated Random Variables

We will sometimes need to get concentration around the sums for *negatively associated random variables*. We will need the following tools.

Definition 10.5.5 (Negatively Associated Random Variables, [DP09, JDP83])**.** *We say that the random variables $X_1, \ldots, X_n$ are* negatively associated *(NA), if any two monotone increasing functions f and g defined on disjoint subsets of the variables in $\{X_i\}_i$ are negatively correlated. That is,*

$$\mathbb{E}[f \cdot g] \leq \mathbb{E}[f] \cdot \mathbb{E}[g].$$

Independent random variables are trivially NA.

Proposition 10.5.6 (0-1 Principle, [DR96])**.** *Let $X_1, \ldots, X_n \in \{0, 1\}$ be binary random variables such that $\sum_i X_i \leq 1$. Then $X_1, \ldots, X_n$ are NA.*

Definition 10.5.7 (Permutation Distribution)**.** *Let $x_1, \ldots, x_n$ be n values and let $X_1, \ldots, X_n$ be random variables taking on all permutations of $(x_1, \ldots, x_n)$ with equal probability. Then we call the collection of random variables $X_1, \ldots, X_n$ a permutation distribution.*

Proposition 10.5.8 ([JDP83])**.** *Collections of random variables that form permutation distributions are NA.*

Proposition 10.5.9 (NA Closure Properties, [DP09])**.**

- **Closure Under Products.** *If $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ are two independent families of random variables that are separately NA, then $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ is also NA.*
- **Disjoint Monotone Aggregation.** *If $X_1, \ldots, X_n$ are NA, and $f_1, \ldots, f_k$ are monotone (either all increasing or all decreasing) functions defined on disjoint subsets of the random variables, then $f_1(X_1, \ldots, X_n), \ldots, f_k(X_1, \ldots, X_n)$ are NA.*

The Chernoff-Hoeffding bounds from Propositions 10.5.1 and 10.5.2 extend to the case where the random variables are NA [DP09].

Bibliography

[ABB+25] Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Martín Costa, Shay Solomon, and Tianyi Zhang. Vizing's theorem in near-linear time. In *57th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2025.

[ABB+26] Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Martín Costa, Shay Solomon, and Tianyi Zhang. Vizing's theorem in deterministic almost-linear time. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2026.

[Arj82] Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with $\Delta+1$ colours. *INFOR: Information Systems and Operational Research*, 20(2):82–101, 1982.

[AS21] Sepehr Assadi and Shay Solomon. Fully dynamic set cover via hypergraph maximal matching: An optimal approximation through a local approach. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 8:1–8:18, 2021.

[Ass25] Sepehr Assadi. Faster Vizing and Near-Vizing Edge Coloring Algorithms. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025.

[BBKO22] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time polylogarithmic in Δ. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 15–25, 2022.

[BBL+25] Aaron Bernstein, Joakim Blikstad, Jason Li, Thatchaphol Saranurak, and Ta-Wei Tu. Combinatorial maximum flow via weighted push-relabel on shortcut graphs. In *66th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2025.

[BBST24] Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, and Ta-Wei Tu. Maximum flow by augmenting paths in $n^{2+o(1)}$ time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2056–2077. IEEE, 2024.

[BCC+24] Sayan Bhattacharya, Din Carmon, Martín Costa, Shay Solomon, and Tianyi Zhang. Faster $(\Delta+1)$-Edge Coloring: Breaking the $m\sqrt{n}$ Time Barrier. In *65th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2024.

[BCF25a] Sayan Bhattacharya, Martín Costa, and Ermiya Farokhnejad. Fully dynamic k-median with near-optimal update time and recourse. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 1166–1177. ACM, 2025.

[BCF+25b] Sayan Bhattacharya, Martin Costa, Ermiya Farokhnejad, Silvio Lattanzi, and Nikos Parotsidis. Almost optimal fully dynamic k-center clustering with recourse. In *Forty-second International Conference on Machine Learning*, 2025.

[BCG+24] Sayan Bhattacharya, Martín Costa, Naveen Garg, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering with fast update time and small recourse. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 216–227. IEEE, 2024.

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *Integer Programming and Combinatorial Optimization (IPCO)*, volume 10328 of *Lecture Notes in Computer Science*, pages 86–98, 2017.

[BCHN18] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic Algorithms for Graph Coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–20. SIAM, 2018.

[BCLP23] Sayan Bhattacharya, Martín Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in $\tilde{o}(k)$ update time. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[BCPS24a] Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Arboricity-Dependent Algorithms for Edge Coloring. In *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 294 of *LIPIcs*, pages 12:1–12:15, 2024.

[BCPS24b] Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Density-Sensitive Algorithms for $(\Delta + 1)$-Edge Coloring. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 23:1–23:18, 2024.

[BCPS24c] Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Nibbling at Long Cycles: Dynamic (and Static) Edge Coloring in Optimal Time. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2024.

[BCSZ25] Sayan Bhattacharya, Martín Costa, Shay Solomon, and Tianyi Zhang. Even Faster $(\Delta + 1)$-Edge Coloring via Shorter Multi-Step Vizing Chains. In *Proceedings of the 2025*

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, pages 4914–4947. SIAM, 2025.

[BD23] Anton Bernshteyn and Abhishek Dhawan. Fast algorithms for vizing's theorem on bounded degree graphs. *CoRR*, abs/2303.05408, 2023.

[BD24] Anton Bernshteyn and Abhishek Dhawan. A linear-time algorithm for $(1+\epsilon)\Delta$-edge-coloring. *arXiv preprint arXiv:2407.04887*, 2024.

[Ber22] Anton Bernshteyn. A fast distributed algorithm for $(\Delta+1)$-edge-coloring. *J. Comb. Theory, Ser. B*, 152:319–352, 2022.

[BGK+22] Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic $(\Delta+1)$-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022.

[BGM17] Sayan Bhattacharya, Manoj Gupta, and Divyarthi Mohan. Improved algorithm for dynamic b-matching. In *25th Annual European Symposium on Algorithms (ESA)*, volume 87 of *LIPIcs*, pages 15:1–15:13, 2017.

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of theACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2830–2842. SIAM, 2021.

[BHNW21] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2537–2549, 2021.

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1872–1885. SIAM, 2019.

[BM76] J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory with Applications*. Macmillan Education UK, 1976.

[BM17] Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *International Conference on Computational Science (ICCS)*, volume 108 of *Procedia Computer Science*, pages 89–98. Elsevier, 2017.

[BMN92] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Inf. Process. Lett.*, 44(5):251–253, 1992.

[Bro11] Daniel G. Brown. How i wasted too long finding a concentration inequality for sums of geometric variables. *Unpublished Manuscript, University of Waterloo*, 2011.

[BSVW24] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Online edge coloring is (nearly) as easy as offline. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 2024.

[BSVW25a] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Deterministic Online Bipartite Edge Coloring. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025.

[BSVW25b] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Online edge coloring: Sharp thresholds. In *66th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2025.

[CCZ25] Shiri Chechik, Hongyi Chen, and Tianyi Zhang. Improved streaming edge coloring. In *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 48:1–48:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[CF25] Martín Costa and Ermiya Farokhnejad. Deterministic k-median clustering in near-optimal time. In *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 62:1–62:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[CH82] Richard Cole and John Hopcroft. On edge coloring bipartite graphs. *SIAM Journal on Computing*, 11(3):540–546, 1982.

[CHL+20] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed Edge Coloring and a Special Case of the Constructive Lovász Local Lemma. *ACM Trans. Algorithms*, 16(1):8:1–8:51, 2020.

[Chr23] Aleksander Bjørn Grodt Christiansen. The Power of Multi-step Vizing Chains. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1013–1026. ACM, 2023.

[Chr26] Aleksander B. G. Christiansen. Deterministic edge colouring. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2026.

[CK24] Julia Chuzhoy and Sanjeev Khanna. Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 83–94. ACM, 2024.

[CKL+25] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *J. ACM*, 72(3):19:1–19:103, 2025.

[COS01] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-Coloring Bipartite Multigraphs in $O(E \log D)$ Time. *Comb.*, 21(1):5–12, 2001.

[CPW19] Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1–25. IEEE Computer Society, 2019.

[DadH90] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Automata, Languages and Programming: 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings 17*, pages 6–19. Springer, 1990.

[DGP98] Devdatt P. Dubhashi, David A. Grable, and Alessandro Panconesi. Near-optimal, distributed edge colouring via the nibble method. *Theor. Comput. Sci.*, 203(2):225–251, 1998.

[DGS25] Aditi Dudeja, Rashmika Goswami, and Michael Saks. Randomized Greedy Online Edge Coloring Succeeds for Dense and Randomly-Ordered Graphs. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025.

[Dha24] Abhishek Dhawan. Edge-coloring algorithms for bounded degree multigraphs. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2120–2157. SIAM, 2024.

[DHZ19] Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[DPS18] Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Trans. Algorithms*, 14(1):8:1–8:35, 2018.

[DR96] Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

[Gal95] Fred Galvin. The list chromatic index of a bipartite multigraph. *Journal of Combinatorial Theory, Series B*, 63(1):153–158, 1995.

[GG24] Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2148–2179. IEEE, 2024.

[GKK10] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in $o(n \log n)$ time in regular bipartite graphs. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 39–46, 2010.

[GNK+85] Harold N Gabow, Takao Nishizeki, Oded Kariv, Daneil Leven, and Osamu Terada. Algorithms for edge coloring. *Technical Rport*, 1985.

[Gol95] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995.

[Got62] C. C. Gotlieb. The construction of class-teacher time-tables. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*, pages 73–77. North-Holland, 1962.

[GP20] Jan Grebik and Oleg Pikhurko. Measurable versions of vizing's theorem. *Advances in Mathematics*, 374(107378), 2020.

[GT89] Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[HLRW24] Monika Henzinger, Jason Li, Satish Rao, and Di Wang. Deterministic near-linear time minimum cut in weighted graphs. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3089–3139. SIAM, 2024.

[Hol81] Ian Holyer. The np-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.

[HP20] Monika Henzinger and Pan Peng. Constant-time dynamic (Δ+1)-coloring. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 53:1–53:18, 2020.

[JDP83] Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, pages 286–295, 1983.

[JMS25] Manuel Jakob, Yannic Maus, and Florian Schager. Towards optimal distributed edge coloring with fewer colors. In *39th International Symposium on Distributed Computing, DISC 2025, October 27-31, 2025, Berlin, Germany*, volume 356 of *LIPIcs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[KLS+22] Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 104–116. ACM, 2022.

[KLS+24] Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab S. Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. *SIAM J. Comput.*, 53(1):87–110, 2024.

[KS87] Howard J Karloff and David B Shmoys. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms*, 8(1):39–52, 1987.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010.

[Kus22] William Kuszmaul. A hash table without hash functions, and how to get the most out of your random bits. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 991–1001. IEEE, 2022.

[Lar12] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 85–94, 2012.

[LL78] Eugene L. Lawler and Jacques Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *J. ACM*, 25(4):612–619, 1978.

[LSH96] Weifa Liang, Xiaojun Shen, and Qing Hu. Parallel algorithms for the edge-coloring and edge-coloring update problems. *J. Parallel Distributed Comput.*, 32(1):66–73, 1996.

[MV80] Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980.

[PD06] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006.

[Pos21] Luke Postle. Graph theory 3-4: Edge coloring multigraphs. YouTube video, 2021.

[PS16] David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016.

[SB24] Mohammad Saneian and Soheil Behnezhad. Streaming edge coloring with asymptotically optimal colors. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 121:1–121:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Sha49] Claude E Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949.

[Sin19] Corwin Sinnamon. Fast and simple edge-coloring algorithms. *arXiv preprint arXiv:1907.03201*, 2019.

[Sol16] Shay Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016.

[SV19] Hsin-Hao Su and Hoa T. Vu. Towards the locality of vizing's theorem. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2019.

[SW18] Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *LIPIcs*, pages 72:1–72:16, 2018.

[vdBCP+23] Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 503–514. IEEE, 2023.

[Viz64] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.

[Viz65] Vadim G Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, 1965.

[Waj24] David Wajc. Personal Communication, Nov. 2024.