Vizing's Theorem in Deterministic Almost-Linear Time

Sepehr Assadi* Soheil Behnezhad† Sayan Bhattacharya‡ Martín Costa§
Shay Solomon¶ Tianyi Zhang‖

Abstract

Vizing's theorem states that any n-vertex m-edge graph of maximum degree Δ can be edge colored using at most $\Delta+1$ different colors. Vizing's original proof is easily translated into a deterministic $O(mn)$ time algorithm. This deterministic time bound was subsequently improved to $\tilde{O}(m\sqrt{n})$ time, independently by [Arjomandi, 1982] and by [Gabow et al., 1985].[1]

A series of recent papers improved the time bound of $\tilde{O}(m\sqrt{n})$ using randomization, culminating in the randomized near-linear time $(\Delta+1)$-coloring algorithm by [Assadi, Behnezhad, Bhattacharya, Costa, Solomon, and Zhang, 2025]. At the heart of all of these recent improvements, there is some form of a sublinear time algorithm. Unfortunately, sublinear time algorithms as a whole almost always require randomization. This raises a natural question: can the deterministic time complexity of the problem be reduced below the $\tilde{O}(m\sqrt{n})$ barrier?

In this paper, we answer this question in the affirmative. We present a deterministic almost-linear time $(\Delta+1)$-coloring algorithm, namely, an algorithm running in $m \cdot 2^{O(\sqrt{\log \Delta})} \cdot \log n = m^{1+o(1)}$ time. Our main technical contribution is to entirely forego sublinear time algorithms. We do so by presenting a new deterministic color-type sparsification approach that runs in almost-linear (instead of sublinear) time, but can be used to color a much larger set of edges.

*University of Waterloo, sepehr@assadi.info
†Northeastern University, s.behnezhad@northeastern.edu
‡University of Warwick, s.bhattacharya@warwick.ac.uk
§University of Warwick, martin.costa@warwick.ac.uk
¶Tel Aviv University, solo.shay@gmail.com
‖Nanjing University, tianyiz25@nju.edu.cn

[1] We use the $\tilde{O}$ notation to hide polylog(n) factors.

Contents

1 Introduction

Let $G = (V, E)$ be a simple and undirected n-vertex m-edge graph with maximum degree Δ. A *proper μ-edge coloring* (shortly, μ-coloring) $\chi : E \to \{1, 2, \ldots, \mu\}$ of G, for an integer parameter $\mu \in \mathbb{N}^+$, is an assignment of colors $\chi(e)$ to each edge $e \in E$, where no two adjacent edges receive the same color. Since the colors of all edges incident on each vertex must be distinct, any proper edge coloring requires at least Δ different colors. While Δ colors always suffice in bipartite graphs, some graphs (e.g., the triangle) require more than Δ colors. Vizing's Theorem asserts that $\Delta + 1$ colors are always sufficient [Viz64]. Furthermore, even for small values of Δ, the problem of determining whether Δ colors suffice is NP-hard [Hol81]. The focus of this work is on the **deterministic** time complexity of $(\Delta + 1)$-coloring in *general* graphs.

Vizing's original proof is easily translated to a deterministic $O(mn)$ time algorithm. This deterministic time bound was improved to $\tilde{O}(m\sqrt{n})$ in the 1980s, independently by [Arj82] and [GNK+85] (more recently [Sin19] proved that the algorithm of [GNK+85] in fact achieves a clean $O(m\sqrt{n})$ time). In *bipartite* graphs, deterministic near-linear time algorithms (even for Δ-coloring) are known since the 80s [CH82, COS01, Alo03, GKK10]. However, in general graphs, the problem is much more challenging. Allowing *randomization*, a sequence of recent papers [Sin19, BCC+24, Ass25, BCSZ25, ABB+25] improved the longstanding time bound of $\tilde{O}(m\sqrt{n})$, culminating in the randomized near-linear time $(\Delta + 1)$-coloring algorithm of [ABB+25] (we discuss these papers in more detail below). Nonetheless, the following fundamental question remains open:

> Can the **deterministic** time complexity of $(\Delta + 1)$-coloring in **general** graphs be reduced below the bound of $\tilde{O}(m\sqrt{n})$ barrier?

In this paper, we present a deterministic almost-linear time $(\Delta + 1)$-coloring algorithm.

Theorem 1.1. *There is a deterministic algorithm that, for any n-vertex m-edge graph $G = (V, E)$ 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, ABB+25] 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 builds on the framework of [ABB+25] and de-randomizes their main "color type reduction" technique, by crucially replacing their sublinear time subroutine with a new "gradual sparsification approach" that we introduce in this paper. 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 [ABB+25]—but can instead apply the required "type reduction" to an almost constant fraction of the graph—as opposed to only $\Theta(1/\Delta)$ fraction in [ABB+25]. We elaborate more on our techniques as well as a technical comparison with prior work in Section 2.

1.1 Previous Work

1.1.1 Recent Randomized Algorithms

As mentioned, the state-of-the-art deterministic runtime for $(\Delta+1)$-coloring is $O(m\sqrt{n})$ due to [Sin19], which provides a more careful analysis of the $\tilde{O}(m\sqrt{n})$ time algorithm of [GNK+85] (see also [Arj82]). Moreover, [Sin19] also presented a much simpler randomized algorithm with the same $O(m\sqrt{n})$ runtime that succeeds with (exponentially) high probability. The first polynomial improvement over this longstanding $\tilde{O}(m\sqrt{n})$ time bound was obtained in two independent works [Ass25] and [BCC+24], presenting two randomized algorithms with the incomparable time bounds of $\tilde{O}(n^2)$ and $\tilde{O}(mn^{1/3})$, respectively. In a follow-up work, the $\tilde{O}(mn^{1/3})$ randomized time bound of [BCC+24] was improved to $\tilde{O}(mn^{1/4})$ time in [BCSZ25]. Finally, a randomized near-linear time algorithm was presented in [ABB+25]; more specifically, the algorithm of [ABB+25] achieves a time bound of $O(m \log \Delta)$ with high probability.

1.1.2 Other Related Work

There is a growing body of work on fast algorithms for edge coloring that use more than $\Delta+1$ colors. A simple *randomized* near-linear time algorithm that uses $\Delta + \tilde{O}(\sqrt{\Delta})$ colors was given already in the 80s [KS87]. More recently, [Ass25] improved these bounds to a $\Delta + O(\log n)$ coloring in $O(m \log \Delta)$ expected time. There are also recent algorithms that run in near-linear (and sometimes linear) time for $(1+\epsilon)\Delta$-coloring [DHZ19, BCPS24c, EK24, BD24, Dha24], for a constant $\epsilon \in (0,1)$; while the algorithm of [EK24] is deterministic, all the others are randomized.

The edge coloring problem has also been studied extensively in restricted graph classes. As mentioned, in bipartite graphs, a Δ-coloring can be computed deterministically in $\tilde{O}(m)$ time [CH82, COS01, Alo03, GKK10]; in particular, a deterministic time bound of $O(m \log \Delta)$ was achieved in [COS01]. In bounded degree graphs, one can deterministically compute a $(\Delta+1)$-coloring in $\tilde{O}(m\Delta)$ time [GNK+85], and a randomized algorithm generalizing this result for bounded arboricity graphs was given in [BCPS24b]; refer also to [BCPS24a, CRV24, Kow24] for additional recent deterministic and randomized algorithms on edge coloring in bounded arboricity graphs. There are also works on edge coloring in subfamilies of bounded arboricity graphs, and in particular in bounded tree-width graphs, planar graphs and bounded genus graphs [CY89, CN90, CK08].

Furthermore, edge coloring has been studied extensively and intensively in various computational models over the past few years, including the dynamic setting [BM17, BCHN18, DHZ19, Chr23, BCPS24c, Chr24], distributed computing [PR01, EPS14, FGK17, GKMU18, GP20, BBKO22, CHL+20, Ber22, Chr23, Dav23], online algorithms [CPW19, BGW21, SW21, KLS+22, BSVW24, BSVW25, DGS25], streaming algorithms [BDH+19, SB24, CMZ24, GS24], and sampling algorithms [ALG22, WZZ24, CWZZ25].

1.2 Organization

In Section 2 we provide a detailed technical overview of our work, which also includes comparison with previous work. In Section 3 we give the necessary notation and preliminaries. In Section 4 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. The rest of the paper is devoted to presenting these subroutines and formally analyzing them by proving these lemmas.

2 Technical Overview

To highlight the main ideas behind our approach, in this section we instantiate our algorithm on bipartite graphs. Let $G = (V, E)$ be the input bipartite graph with maximum degree Δ. Consider any **partial Δ-coloring** $\chi : E \longrightarrow [\Delta] \cup \{\perp\}$ of G, where $\{e \in E : \chi(e) = \perp\}$ is the set of **uncolored** edges. We say that χ is **valid** iff no two edges incident on the same vertex receive the same color (from $[\Delta]$) under χ. Unless explicitly stated otherwise, we will only consider valid partial colorings. Consider any subset $S \subseteq \{e \in E : \chi(e) = \perp\}$ of uncolored edges. The phrase **extending the coloring χ to S** refers to the following operation: Change the colors (under χ) of some edges in $\{e \in E : \chi(e) \neq \perp\}$ and assign a color $\chi(e) \in [\Delta]$ to every edge $e \in S$, while ensuring that χ remains a valid partial coloring of G.

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 2.1. *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 valid 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 2.1 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 4.1 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 2.2. *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 2.1, 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 3.2). Thus, although Theorem 2.1 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 2.1 contains all the key conceptual insights that eventually lead to Theorem 1.1. Second, we use Assumption 2.2 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 2.1, we introduce some key notations and concepts. Section 2.2 presents Theorem 2.3, which summarizes a *type sparsification* framework that underpins our entire algorithm. Immediately after stating Theorem 2.3, we explain how it implies Theorem 2.1. In Section 2.3, we present a *randomized* algorithm for type sparsification. In Section 2.4, we compare this new randomized algorithm against the algorithm of [ABB+25], and point out the barrier towards efficiently derandomizing the latter. Section 2.5 demonstrates how the new randomized algorithm from Section 2.3 helps us overcome this barrier. Specifically, we derandomize the algorithm from Section 2.3, which leads us to the proof of Theorem 2.3.

2.1 Notations and Preliminaries

Let $C = \{1, \ldots, \Delta\}$ be the palette of available colors. Moreover, let $\mathsf{miss}_\chi(u) := \{\alpha \in C : \chi(u, v) \neq \alpha \text{ for all } (u, v) \in E\}$ denote the set of **missing colors** at a vertex $u \in V$, w.r.t. χ. 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** $\tau \in A \times B$.

2.2 The Framework: Type Sparsification

The theorem below encapsulates our main technical contribution in this paper.

Theorem 2.3. *Consider any palette C of colors, and any valid 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 $E \backslash U$, 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 2 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 2.3 holds even if $|C| < \Delta$, as long as $\chi : E \longrightarrow C \cup \{\perp\}$ remains a valid partial coloring.

We say that Theorem 2.3 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 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 2.3 is to modify the partial coloring χ so that this matrix looks more diagonal. See Figure 1 for an illustration.

We now explain why Theorem 2.3 implies Theorem 2.1. 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]

[2]We only make this assumption in the technical overview to simplify the technical details, since this leads to a

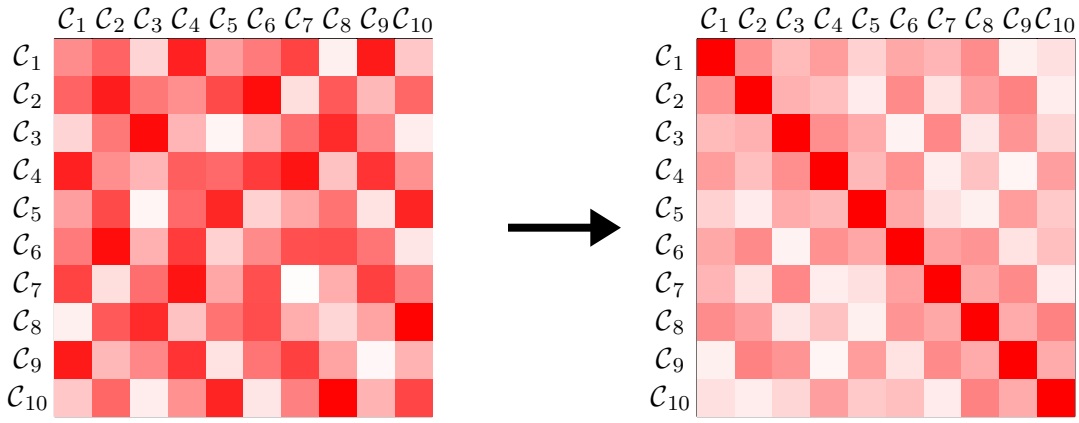

Figure 1: In this example, we have $\eta = 10$. The matrix heatmap is initially the left one. After we run the algorithm of Theorem 2.3, most weights are concentrated along the diagonal.

Next, we apply the type sparsification algorithm from Theorem 2.3 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 2.3 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 2.3 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 $\eta = \Delta^\epsilon$ 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

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.

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

$\tilde{O}(|\mathcal{E}'|)$ 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 2.1 follows if we balance the relevant terms by setting $\epsilon := \Theta(1/\sqrt{\log \Delta})$.

2.3 A Randomized Algorithm for Type Sparsification

We now present a *randomized* algorithm that performs the same task as specified in the statement of Theorem 2.3. Theorem 2.15 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 2.3). 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 2.3 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 2.2). 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 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., $\phi_{i \to j}^{-1} = \phi_{j \to i}$. 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 2 for an illustration.

Observation 2.4. *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. □

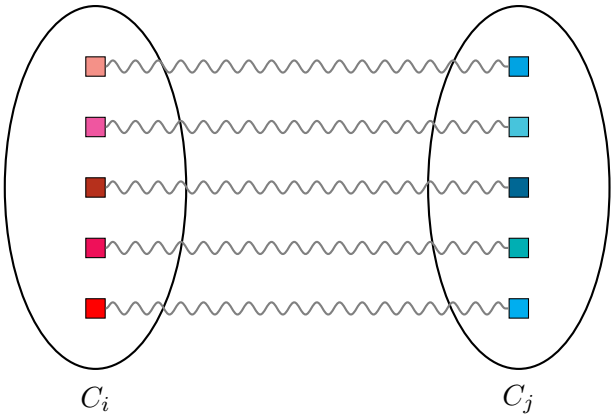

Figure 2: The mapping $\phi_{i\to j}$ defines a matching between colors in C_i, C_j. In this example we have drawn 5 colors from color sets C_i, C_j, and colors connected by strings are matched together.

2.3.1 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 2.5. *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 2.5, 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 2.5 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

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

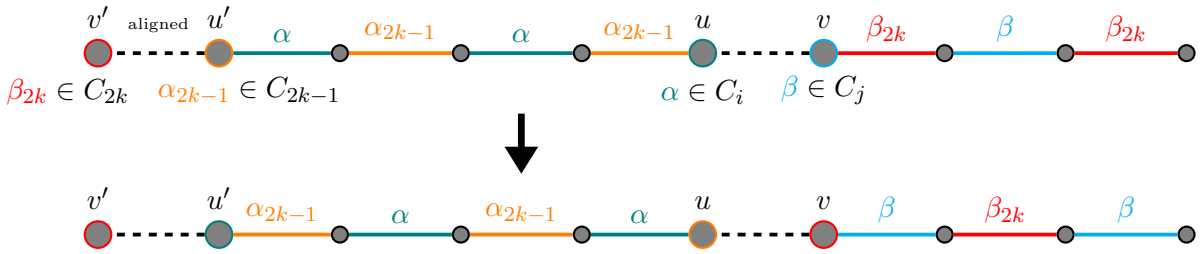

Figure 3: In this example, we attempt to align edge (u, v) by flipping the $\{\alpha, \alpha_{2k-1}\}$-alternating path $P_u(e)$ from u and the $\{\beta, \beta_{2k}\}$-alternating path $P_v(e)$ from v. However, flipping the $\{\alpha, \alpha_{2k-1}\}$-alternating path from u damages a previously aligned edge (u', v') as u' will not miss color α_{2k-1} anymore. In this case, (u', v') is responsible for (u, v).

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 2.6). **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 3 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 (u, v) to the set U_k, and then proceed towards the next iteration.

2.3.2 Analysis

From the description in Section 2.3.1, 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 2.3. It remains to bound the expected time complexity of this algorithm. Towards this end, we start with a couple of simple observations. Subsequently, Claim 2.8 and Corollary 2.9 bound the expected runtime of a given iteration within a round $k \in [\eta]$, whereas Claim 2.10 and Corollary 2.13 bound the probability that a given iteration is successful. Putting everything together, Lemma 2.14 bounds the expected runtime of a given round. This, in turn, leads us to Theorem 2.15.

Observation 2.6. *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 2.3.1. □

Observation 2.7. *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 2.5, 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 2.8. *At any given point in time within a round $k \in [\eta]$, define*

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

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

Proof. Recall that the edge-set $U^\star \setminus (U_1 \cup \cdots \cup U_k) \subseteq U$ forms a matching (see Theorem 2.3). By Observation 2.4, 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 2.2). Using an analogous argument, we get $\sum_{P \in \mathcal{P}_{\texttt{right}}} |P| = O(m)$. This concludes the proof. □

Corollary 2.9. *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 2.3). 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

$$\mathbb{E}\left[|P_u(e)|\right] = \frac{\sum_{P \in \mathcal{P}_{\texttt{left}}} |P|}{|U^\star \setminus (U_1 \cup \cdots \cup U_k)|} \text{ and } \mathbb{E}\left[|P_v(e)|\right] = \frac{\sum_{P \in \mathcal{P}_{\texttt{right}}} |P|}{|U^\star \setminus (U_1 \cup \cdots \cup U_k)|}. \tag{1}$$

The corollary now follows from Observation 2.7 and Claim 2.8. □

Claim 2.10. *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 2.11. *Each edge $e' \in U_1 \cup \cdots \cup U_{k-1}$ is responsible for at most 4 bad edges.*

Observation 2.12. *Each edge $e' \in U_k$ is responsible for at most 4η bad edges.*

Taken together, Observation 2.11 and Observation 2.12 (along with Invariant 2.5) 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 2.7, *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.

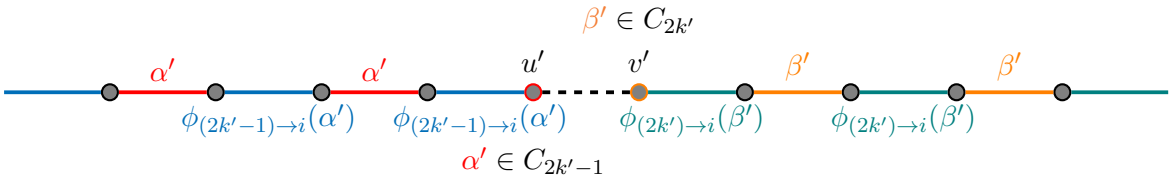

Figure 4: In this picture, $e' = (u', v') \in U_{k'}$, for some $k' < k$. We have drawn two alternating paths of types $\{\alpha', \phi_{(2k'-1)\to i}(\alpha')\}$ and $\{\beta', \phi_{(2k')\to i}(\beta')\}$ from u', v' which could damage some edges $e^\star$.

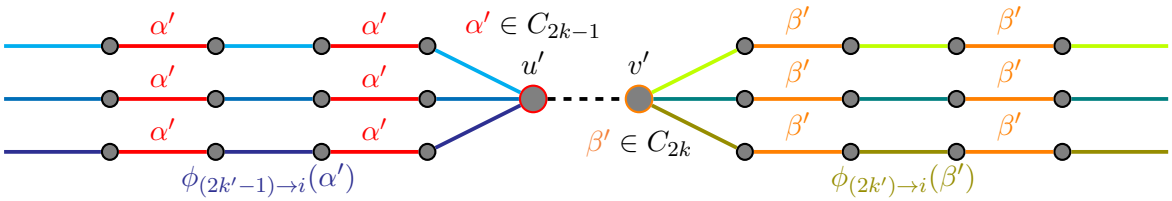

Figure 5: In this picture, $e' = (u', v') \in U_k$. We have drawn some alternating paths of types $\{\alpha', \phi_{(2k'-1)\to i}(\alpha')\}$ and $\{\beta', \phi_{(2k')\to i}(\beta')\}$ from u', v' for three choices of $i \in [2\eta] \setminus \{2k-1, 2k\}$.

For Observation 2.11, 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 2.5). 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 $i \in \{2k-1, 2k\}$. So, the edge e' can be responsible for at most $2 \times 2 = 4$ bad edges. Check Figure 4 for an illustration.

For Observation 2.12, 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 2.5). 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 5 for an illustration. □

Remark. Note that the proofs of Observation 2.7, Claim 2.8 and Claim 2.10 do not rely on randomness in any way; we will use these statements also in the deterministic setting of Section 2.5.

Corollary 2.13. *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 2.10. □

Lemma 2.14. *In the algorithm from Section 2.3.1, each round takes $\tilde{O}(m/\eta)$ expected time.*

Proof. By Corollary 2.9, each iteration within a round takes $\tilde{O}(m/\lambda)$ expected time. By Corollary 2.13, 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 2.15. *The algorithm from Section 2.3.1 performs the same task as specified in the statement of Theorem 2.3, and runs in expected $\tilde{O}(m)$ time.*

Proof. Since the algorithm consists of η rounds, the theorem follows from Lemma 2.14. □

2.4 Comparison with the Randomized Algorithm of [ABB+25]

We now explain how to derive the result of [ABB+25] from the randomized algorithm presented in Section 2.3. 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 2.3.1. By Lemma 2.14, this takes $\tilde{O}(m/\eta) = \tilde{O}(n\Delta/\eta) = \tilde{O}(n)$ *expected time*. By Invariant 2.5, 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, [ABB+25] 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 2.2), 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 2.3.1 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 the algorithm of [ABB+25]. 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 algorithm from [ABB+25] in a way that is more compatible with our framework discussed in Section 2.3, 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, de-randomizing 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 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.

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

Our main conceptual contribution in this paper is to formulate the notion of *type sparsification*, as captured in the statement of Theorem 2.3, and derive the new randomized algorithm presented in Section 2.3. Since Theorem 2.3 asks for a time complexity of $\tilde{O}(m \cdot \text{poly}(\eta))$ and the randomized algorithm from Section 2.3 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 2.14. In Section 2.5, 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 2.3, which, as explained in Section 2.2, implies Theorem 2.1.

2.5 Derandomization: Proof of Theorem 2.3

We now show how to derandomize the algorithm from Section 2.3. Recall the description of the randomized algorithm from Section 2.3.1. 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 2.3.1 consists of η rounds, this leads us to Theorem 2.3.

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 2.16, 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 2.17, 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 2.18 and Claim 2.19 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 2.3), 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 2.8.

To summarize, what we have described above is a deterministic procedure that runs in $\tilde{O}(m)$

[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.

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 2.5). 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 2.3, since the algorithm from Section 2.3.1 consists of η rounds.

Claim 2.16. *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 2.7 and Claim 2.10, 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 2.17. *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 2.8. □

Claim 2.18. *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 $e = (u, v) \in B$ 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 2.3.1, 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 2.19. *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 2.3.1, 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 $\tau \cap \tau' = \emptyset$. □

3 Preliminaries

In this section, we give the preliminaries and define the notation used throughout the paper. The notation and preliminaries of our paper are very similar to [ABB+25].

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 with colors C. We refer to edges $e \in E$ with $\chi(e) = \perp$ as *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 collections in Section 3.2) will be defined with respect to this particular partial coloring χ.

3.2 U-Fans and Separable Collections

We begin by defining the notion of *u-fans* that was used by [GNK+85].[8]

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

Definition 3.1 (u-fan, [GNK+85]). *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 3.1.

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: Following the approach of [ABB+25], throughout this paper, we often 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'. We will be particularly interested in collections satisfying the following useful property, which we refer to as *separability*.

Definition 3.2 (Separable Collection, [ABB+25]). *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.*

As discussed in [ABB+25], the second property of this definition is rather important since we need to ensure that different u-fans are not interfering with each other when they share common vertices. See Figure 6 for an illustration.

3.3 Constructing Separable Collections

To construct separable collections of u-fans, we use a fast deterministic algorithm given by [ABB+25] 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. More specifically, we use the following lemma of [ABB+25] as a black box.

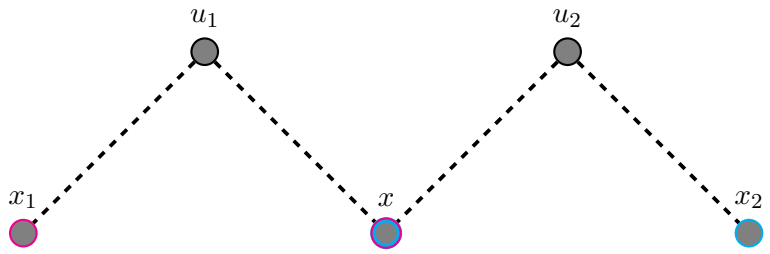

Figure 6: This picture shows two u-fans $(u_1, x_1, x, \cdot, \beta_1, \beta_1)$ and $(u_2, x_2, x, \cdot, \beta_2, \beta_2)$ sharing a common vertex x. The separability condition requires that $\beta_1 \neq \beta_2$; for instance, β_1 and β_2 could be magenta and cyan as shown here.

Lemma 3.3 ([ABB+25], Lemma 6.2)**.** *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 3.3 crucially uses Vizing fans and chains. The rest of our algorithm does not use Vizing fans and chains.

3.4 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 1 gives a formal description of how we implement the procedure.

Algorithm 1: `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 3.4 ([ABB+25], Lemma 5.4)**.** *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.*

3.5 Data Structures

In Section 8, we describe the data structures that we use to implement our algorithm. Our data structures are simple and almost identical to the data structures of [ABB+25], 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.

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.

- $\textsc{Insert}_{\mathcal{U}}(\boldsymbol{f})$: 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.
- $\textsc{Delete}_{\mathcal{U}}(\boldsymbol{f})$: 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.
- $\textsc{Find-U-Fan}_{\mathcal{U}}(x, c)$: 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.
- $\textsc{Missing-Color}_{\mathcal{U}}(x)$: The input to this query is a vertex $x \in V$. In response, the data structure returns an arbitrary color from the set $\mathsf{miss}_{\chi}(x) \setminus C_{\mathcal{U}}(x)$.[9]

The following claim shows that it is always possible to answer a Missing-Color query.

Claim 3.5. *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.

- $\textsc{Initialize}(G, \chi)$: 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, 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.**

4 The Main Algorithm (Proof of Theorem 1.1)

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.

The first of these subroutines and the main technical component of our algorithm is a subroutine called `Sparsify-Types`, which is described in Section 5. The following lemma (proved in Section 5) summarizes the behavior of `Sparsify-Types`.

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

Lemma 4.1. *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 $10 \le \eta \le \mu/10$, 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 $\mathcal{C}_1, \dots, \mathcal{C}_\eta \subseteq C$ with the following properties:*

1. *For all $k \in [\eta]$, we have that $|\mathcal{C}_k| \le \mu/\eta$.*
2. *The u-fans in $\mathcal{U}$ have types in $\bigcup_{k=1}^{\eta}(\mathcal{C}_k \times \mathcal{C}_k)$ and $|\mathcal{U}| \ge \lambda/100$.*

Furthermore, the algorithm is deterministic and runs in time $O(m\eta^4 \log \mu)$.

The second subroutine, described in Section 6, is called `Color-Small`, and we use it to extend the coloring to sufficiently small subgraphs. The following lemma (proved in Section 6) summarizes the behavior of `Color-Small`.

Lemma 4.2. *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 7, 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 7) summarizes the behavior of `Extend-Coloring`.

Lemma 4.3. *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 $\eta \ge 10$, 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 4.3, we can now prove Theorem 1.1.

4.1 Proof of Theorem 1.1

We begin by proving the following corollary of Lemma 4.3, which we use to efficiently extend a $(\Delta+1)$-coloring of a graph G.

Corollary 4.4. *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 3.3, 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 4.3 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 $\Omega(1/100^{\log \Delta/\log \eta+1}) \ge \Omega(1/100^{\sqrt{\log_2 \Delta}})$ 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 $O(m \cdot 2^{4\sqrt{\log_2 \Delta}} \log^2 \Delta) \le O(m \cdot 2^{6\sqrt{\log_2 \Delta}})$ time, since we repeat this process for $O(100^{\sqrt{\log_2 \Delta}} \log \lambda) \le O(2^{7\sqrt{\log_2 \Delta}} \log n)$ iterations, this takes $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n)$ time in total. □

We prove Theorem 1.1 by applying Corollary 4.4 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 4.4 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 $O(m \cdot 2^{13\sqrt{\log_2 \Delta}} \log n \log \Delta) \leq O(m \cdot 2^{14\sqrt{\log_2 \Delta}} \log n)$.

5 The Algorithm `Sparsify-Types` (Proof of Lemma 4.1)

In this section, we describe and analyze the subroutine `Sparsify-Types`, proving Lemma 4.1, which we restate below.

Lemma 4.1. *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 $O(m\eta^4 \log \mu)$.

5.1 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 $\tau(\boldsymbol{f}) \in [q] \times [q]$.[10] 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

[10] Recall that $[q] \times [q]$ denotes the set of *unordered* subsets $\{\{c, c'\} \mid c, c' \in [q]\}$.

Claim 5.2. 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 $|\hat{\mathcal{U}}| = \Omega(\lambda)$.

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 $\{P_u, P_v, P_w\}$.[11] 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 2 provides the pseudocode for the algorithm `Compute-Paths` that computes the set of k-relevant alternating paths of $\boldsymbol{f}$.

Algorithm 2: `Compute-Paths`$(\boldsymbol{f}, k)$

```
Let f = (u, v, w, c_f(u), c_f(v), c_f(w))
Let c_f(u) = C_i(j) and c_f(v) = c_f(w) = C_{i'}(j'), where i, i' ∈ [2η] and j, j' ∈ [r]
Let ℓ ← 2k − 1 and ℓ' ← 2k
if i = 2k or i' = 2k − 1 then
    Let ℓ ← 2k and ℓ' ← 2k − 1
Let P_u be the {C_ℓ(j), C_i(j)}-alternating path starting at u
Let P_v and P_w be the {C_{ℓ'}(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.

[11] 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.

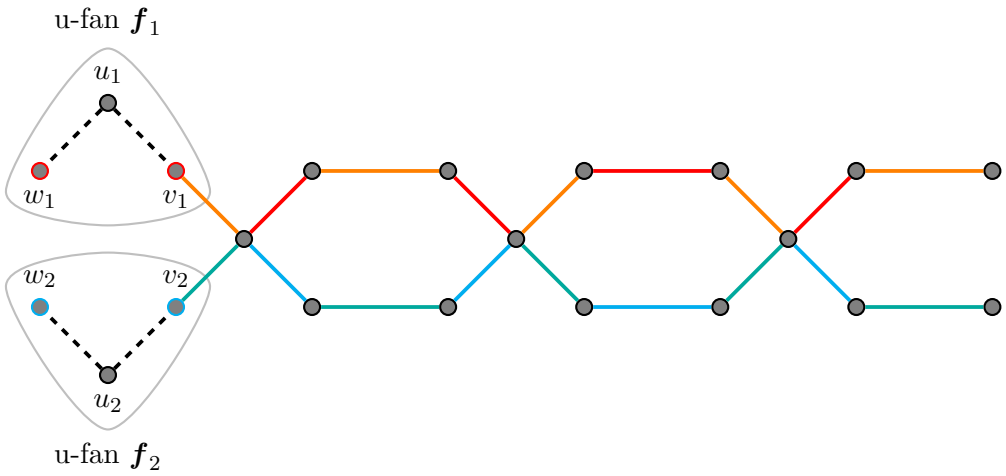

Figure 7: In this picture we have two different u-fans $\boldsymbol{f}_1 = (u_1, v_1, w_1, \cdot, C_i(j_1), C_i(j_1))$ and $\boldsymbol{f}_2 = (u_2, v_2, w_2, \cdot, C_i(j_2), C_i(j_2))$. When $j_1 \neq j_2$, we have $\{C_i(j_1), C_{2k}(j_1)\} \cap \{C_i(j_2), C_{2k}(j_2)\} = \emptyset$. Then two k-relevant 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.

Claim 5.1. *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 5.1 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 $\mathcal{V}$ so that they are all contained in $C_{2k-1} \times C_{2k} \subseteq \mathcal{C}_k \times \mathcal{C}_k$, making them social.[12] See Figure 7 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 3.

Algorithm 3: `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

[12] 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.

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.

5.2 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 4 gives a formal description of the algorithm. We note that, throughout the run of Algorithm 4, 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 4 and when we remove damaged u-fans from $\mathcal{U}$ in Line 4. 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 4, these are defined with respect to the *current state* of the separable collection $\mathcal{U}$. In Section 5.4, we show how to compute these subsets efficiently.

Algorithm 4: `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 5.3, 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 4.1. In Section 5.4, we show how to implement the algorithm `Sparsify-Types` to run in time $O(m\eta^4 \log \mu)$.

5.3 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 5.2. *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 5.3. *Consider some iteration of Algorithm 4 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 4. 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 5.4. *At the start of each iteration of Algorithm 4, we have that $|\mathcal{U}| \geq \lambda/2$.*

Proof. It follows from Claim 5.2 that Line 4 of Algorithm 4 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 5.3 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 5.5. *During each iteration of Algorithm 4, 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 $\lambda/(4\eta^2)$.

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

$$\mathcal{U} \setminus \mathcal{B}_{k^\star} = \bigcup_{1 \leq i < i' \leq \eta} (\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}). \tag{2}$$

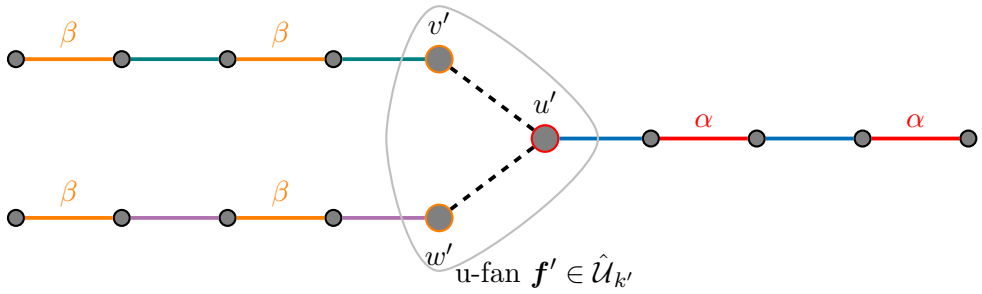

Figure 8: In this example, $\boldsymbol{f}' = (u', v', w', \alpha, \beta, \beta) \in \hat{\mathcal{U}}_{k'}$, where $k' \neq k$ and $\alpha = C_{2k'-1}(j_1), \beta = C_{2k'}(j_2)$. Then, for any $i \in \{2k-1, 2k\}$, u' could be the endpoint of a k-relevant path of type $\{\alpha = C_{2k'-1}(j_1), C_i(j_1)\}$, 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.

By a simple averaging argument and Equation (2), we get that

$$|\mathcal{U}_{i^\star,i'^\star} \setminus \mathcal{B}_{k^\star}| \geq \frac{1}{\eta^2} \cdot \sum_{1 \leq i < i' \leq \eta} |\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}| = \frac{1}{\eta^2} \cdot |\mathcal{U} \setminus \mathcal{B}_{k^\star}|. \tag{3}$$

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 5.6. *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 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, 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 9 for an illustration.

Thus, accounting for the social u-fans, the total number of k-bad u-fans is at most

$$6 \cdot |\hat{\mathcal{U}} \setminus \hat{\mathcal{U}}_k| + 6\eta \cdot |\hat{\mathcal{U}}_k| + |\hat{\mathcal{U}}| \leq 7 \cdot |\hat{\mathcal{U}}| + 6\eta \cdot |\hat{\mathcal{U}}_k|. \qquad \square$$

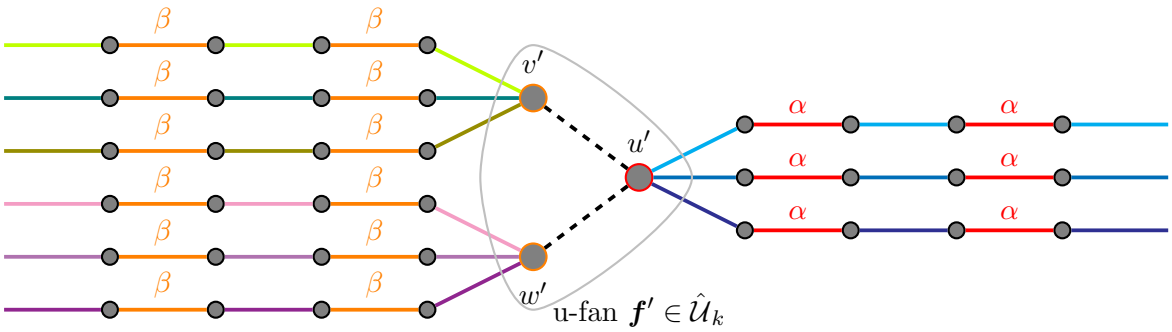

Figure 9: In this example, $\boldsymbol{f}' = (u', v', w', \alpha, \beta, \beta) \in \hat{\mathcal{U}}_k$, where $\alpha = C_{2k-1}(j_1), \beta = C_{2k}(j_2)$. Then, for any $i \in [2\eta] \setminus \{2k-1, 2k\}$, u' could be the endpoint of a k-relevant path of type $\{\alpha = C_{2k-1}(j_1), C_i(j_1)\}$, 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 $6\eta - 6$.

It follows from Claim 5.6 that $|\mathcal{B}_{k^\star}| \leq 7 \cdot \lambda/100 + 6\eta \cdot \lambda/(100\eta) \leq \lambda/4$. Since, by Claim 5.4, the size of $\mathcal{U}$ is at least $\lambda/2$, it follows from Equation (3) 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 5.3 and 5.5, we get the following corollary, which upper bounds the number of iterations performed by the algorithm.

Corollary 5.7. *During each iteration of Algorithm 4, 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 $\bigcup_{k=1}^{\eta} \mathcal{C}_k \times \mathcal{C}_k$.

5.4 Implementation of `Sparsify-Types`

In this section, we show how to implement each iteration of Algorithm 4 in $O(m\eta^2 \log \mu)$ time. By the analysis in Section 5.3, 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 $\{\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}\}_{1\leq i<i'\leq\eta}$ in $O(m\eta^2 \log \mu)$ time and then find the values $i^\star, i'^\star = \arg\max_{1\leq i<i'\leq\eta} |\mathcal{U}_{i,i'} \setminus \mathcal{B}_{k^\star}|$.

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 5.6, 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.

6 The Algorithm `Color-Small` (Proof of Lemma 4.2)

In this section, we describe and analyze the subroutine `Color-Small`, proving Lemma 4.2, which we restate below.

Lemma 4.2. *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 $O(m\mu^2 \log \mu)$. □

7 The Algorithm `Extend-Coloring` (Proof of Lemma 4.3)

In this section, we describe and analyze the subroutine `Sparsify-Types`, proving Lemma 4.3, which we restate below.

Lemma 4.3. *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* $O(m\eta^4\log^2(\mu)/\log(\eta))$.

7.1 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 4.1) 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 $\chi_1, \ldots, \chi_{\eta+1}$.

7.2 Analysis of `Extend-Coloring`

We prove Lemma 4.3 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 4.2 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 4.3 holds whenever $\mu \leq 10\eta$.

For the induction step, suppose that Lemma 4.3 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 $\chi_k^\star$ 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 $\chi_{\eta+1}^\star$ 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 $\chi_1^\star, \ldots, \chi_{\eta+1}^\star$, which we denote by $\chi^\star$.

The following claim shows that these subproblems are all feasible.

Claim 7.1. *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 7.1 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 4.1, 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 4.1. 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 7.1, 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 4.1. 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 4.2 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 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 3.5.

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) 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 $\mathsf{miss}_\chi(u) \cap C[\deg_G(u) + 1]$.[13]

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 $C_{\mathcal{U}}(u) := \{c_{\boldsymbol{f}}(u) \mid \boldsymbol{f} \in \mathcal{U}, u \in \boldsymbol{f}\}$.

[13] 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 $O(m)$.

8.1 Implementing the Operations from Section 3.5

We now describe how to implement each of the operations from Section 3.5.

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 $\overline{C}_{\mathcal{U}}(x)$.

Acknowledgements

Sepehr Assadi is supported in part by a Sloan Research Fellowship, an NSERC Discovery Grant (RGPIN-2024-04290), and a Faculty of Math Research Chair grant from University of Waterloo.

Soheil Behnezhad is funded by an NSF CAREER award CCF-2442812 and a Google Faculty Research Award. Martín Costa is supported by a Google PhD Fellowship. Shay Solomon is funded by the European Union (ERC, DynOpt, 101043159). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Shay Solomon is also funded by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF). Work of Tianyi Zhang was done while at ETH Zürich when supported by funding from the starting grant "A New Paradigm for Flow and Cut Algorithms" (no. TMSGI2_218022) of the Swiss National Science Foundation.

References

[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.

[ALG22] Dorna Abdolazimi, Kuikui Liu, and Shayan Oveis Gharan. A matrix trickle-down theorem on simplicial complexes and applications to sampling colorings. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 161–172. IEEE, 2022.

[Alo03] Noga Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Information Processing Letters*, 85(6):301–302, 2003.

[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.

[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.

[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.

[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.

[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 (Δ + 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.

[BD24] Anton Bernshteyn and Abhishek Dhawan. A linear-time algorithm for $(1 + \epsilon)\Delta$-edge-coloring. *arXiv preprint arXiv:2407.04887*, 2024.

[BDH+19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *27th Annual European Symposium on Algorithms (ESA)*, volume 144 of *LIPIcs*, pages 15:1–15:14, 2019.

[Ber22] Anton Bernshteyn. A fast distributed algorithm for $(\Delta + 1)$-edge-coloring. *J. Comb. Theory, Ser. B*, 152:319–352, 2022.

[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.

[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.

[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.

[BSVW25] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Deterministic Online Bipartite Edge Coloring. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 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.

[Chr24] Aleksander B. G. Christiansen. Deterministic dynamic edge-colouring. *CoRR*, abs/2402.13139, 2024.

[CK08] Richard Cole and Łukasz Kowalik. New linear-time algorithms for edge-coloring planar graphs. *Algorithmica*, 50(3):351–368, 2008.

[CMZ24] Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. Streaming edge coloring with subquadratic palette size. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 40:1–40:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[CN90] Marek Chrobak and Takao Nishizeki. Improved edge-coloring algorithms for planar graphs. *Journal of Algorithms*, 11(1):102–116, 1990.

[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)*, pages 1–25. IEEE Computer Society, 2019.

[CRV24] Aleksander B. G. Christiansen, Eva Rotenberg, and Juliette Vlieghe. Sparsity-parameterised dynamic edge colouring. In *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 294 of *LIPIcs*, pages 20:1–20:18, 2024.

[CWZZ25] Zejia Chen, Yulin Wang, Chihao Zhang, and Zihan Zhang. Decay of correlation for edge colorings when $q > 3\Delta$. *arXiv preprint arXiv:2502.06586*, 2025.

[CY89] Marek Chrobak and Moti Yung. Fast algorithms for edge-coloring planar graphs. *Journal of Algorithms*, 10(1):35–51, 1989.

[Dav23] Peter Davies. Improved distributed algorithms for the lovász local lemma and edge coloring. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4273–4295. SIAM, 2023.

[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. A simple algorithm for near-vizing edge-coloring in near-linear time. *arXiv preprint arXiv:2407.16585*, 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.

[EK24] Michael Elkin and Ariel Khuzman. Deterministic Simple $(1 + \epsilon)$-Edge-Coloring in Near-Linear Time. *arXiv preprint arXiv:2401.10538*, 2024.

[EPS14] Michael Elkin, Seth Pettie, and Hsin-Hao Su. $(2\Delta - 1)$-Edge-Coloring is Much Easier than Maximal Matching in the Distributed Setting. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 355–370. SIAM, 2014.

[FGK17] Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–191. IEEE, 2017.

[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.

[GKMU18] Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic distributed edge-coloring with fewer colors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 418–430, 2018.

[GNK+85] Harold N Gabow, Takao Nishizeki, Oded Kariv, Daneil Leven, and Osamu Terada. Algorithms for edge coloring. *Technical Report*, 1985.

[GP20] Jan Grebik and Oleg Pikhurko. Measurable versions of vizing's theorem. *Advances in Mathematics*, 374(107378), 2020.

[GS24] Prantar Ghosh and Manuel Stoeckl. Low-memory algorithms for online edge coloring. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 71:1–71:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Hol81] Ian Holyer. The np-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.

[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.

[Kow24] Lukasz Kowalik. Edge-Coloring Sparse Graphs with Δ Colors in Quasilinear Time. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 81:1–81:17, 2024.

[KS87] Howard J Karloff and David B Shmoys. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms*, 8(1):39–52, 1987.

[PR01] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.

[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.

[Sin19] Corwin Sinnamon. Fast and simple edge-coloring algorithms. *arXiv preprint arXiv:1907.03201*, 2019.

[SW21] Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198 of *LIPIcs*, pages 109:1–109:18, 2021.

[Viz64] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.

[WZZ24] Yulin Wang, Chihao Zhang, and Zihan Zhang. Sampling Proper Colorings on Line Graphs Using $(1+o(1))\Delta$ Colors. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1688–1699, 2024.