Density-Sensitive Algorithms for $(\Delta+1)$-Edge Coloring

Sayan Bhattacharya
University of Warwick, UK

Martín Costa
University of Warwick, UK

Nadav Panski
Tel Aviv University, Israel

Shay Solomon
Tel Aviv University, Israel

Abstract

Vizing's theorem asserts the existence of a $(\Delta+1)$-*edge coloring* for any graph G, where $\Delta = \Delta(G)$ denotes the maximum degree of G. Several polynomial time $(\Delta+1)$-edge coloring algorithms are known, and the state-of-the-art running time (up to polylogarithmic factors) is $\tilde{O}(\min\{m\sqrt{n}, m\Delta\})$,[1] by Gabow, Nishizeki, Kariv, Leven and Terada from 1985, where n and m denote the number of vertices and edges in the graph, respectively. Recently, Sinnamon shaved off a polylog(n) factor from the time bound of Gabow et al.

The *arboricity* $\alpha = \alpha(G)$ of a graph G is the minimum number of edge-disjoint forests into which its edge set can be partitioned, and it is a measure of the graph's "uniform density". While $\alpha \leq \Delta$ in any graph, many natural and real-world graphs exhibit a significant separation between α and Δ.

In this work we design a $(\Delta+1)$-edge coloring algorithm with a running time of $\tilde{O}(\min\{m\sqrt{n}, m\Delta\}) \cdot \frac{\alpha}{\Delta}$, thus improving the longstanding time barrier by a factor of $\frac{\alpha}{\Delta}$. In particular, we achieve a near-linear runtime for bounded arboricity graphs (i.e., $\alpha = \tilde{O}(1)$) as well as when $\alpha = \tilde{O}(\frac{\Delta}{\sqrt{n}})$. Our algorithm builds on Gabow et al.'s and Sinnamon's algorithms, and can be viewed as a density-sensitive refinement of them.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph Algorithms, Edge Coloring, Arboricity

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.23

Related Version *Full Version*: https://arxiv.org/abs/2307.02415

Funding 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 and Nadav Panski are supported by the Israel Science Foundation (ISF) grant No.1991/1. Shay Solomon is also supported by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF).

1 Introduction

A *(proper) k(-edge) coloring* in a graph G is a coloring of edges, where each edge is assigned a color from the set $[k] := \{1, \ldots, k\}$, such that no two adjacent edges have the same color. Clearly, k must be at least as large as the maximum degree $\Delta = \Delta(G)$ of the graph G, and

[1] Here and throughout the $\tilde{O}$ notation suppresses polylog(n) factors.

Vizing's theorem [22] states that $\Delta + 1$ colors always suffice; in some graphs $\Delta + 1$ colors are necessary.

Several polynomial time $(\Delta+1)$-coloring algorithms are known, including a simple $O(mn)$-time algorithm by Misra and Gries [18], which is a simplification of an earlier algorithm by Bollobás [5]. In 1985 Gabow, Nishizeki, Kariv, Leven and Terada [14] presented a $(\Delta + 1)$-coloring algorithm with a running time of $O(\min\{m\sqrt{n \log n}, m\Delta \log n\})$. A recent work by Sinnamon [21] shaves off some polylog(n) factors; specifically, Sinnamon removed the $\sqrt{\log n}$ factor from the term $m\sqrt{n \log n}$, to achieve a clean runtime bound of $O(m\sqrt{n})$. Nonetheless, up to the polylog(n) factors, no improvement on the runtime of the algorithm of [14] was reported to date. We summarize this state-of-the-art result in the following theorem:

▶ **Theorem 1** (Gabow et al. [14]). *For any n-vertex m-edge graph of maximum degree Δ, a $(\Delta + 1)$-edge coloring can be computed within time $\tilde{O}(\min\{m\sqrt{n}, m\Delta\})$.*

Note that the runtime bound provided by Theorem 1 is near-linear in bounded degree graphs. However, in most graphs of interest, the maximum degree Δ is large. The question of whether or not one can significantly improve this runtime bound in graphs of large maximum degree Δ has remained open.

Bounded Arboricity. Sparse graphs, or graphs of "low density", are of importance in both theory and practice. A key definition that captures the property of low density in a "uniform manner" is bounded *arboricity*, which constrain the average degree of any subgraph.

▶ **Definition 2.** *Graph G has* arboricity *$\alpha = \alpha(G)$ if $\frac{m_s}{n_s - 1} \le \alpha$, for every $S \subseteq V$, where m_s and n_s are the number of edges and vertices in the graph induced by S, respectively. Equivalently (by the Nash-Williams theorem [19]), the edges of a graph of arboricity α can be decomposed into α edge-disjoint forests.*

While $\alpha \le \Delta$ holds in any graph G, there might be a large separation between α and Δ; e.g., for the n-star graph we have $\alpha = 1, \Delta = n - 1$. A large separation between α and Δ is exhibited in many natural and real-world graphs, such as the world wide web graph, social networks and transaction networks, as well as in various random distribution models, such as the preferential attachment model. Note also that the family of bounded arboricity graphs, even for $\alpha = O(1)$, includes all graphs that exclude a fixed minor, which, in turn include all bounded treewidth and bounded genus graphs, and in particular all planar graphs.

In this work we present a near-linear time $(\Delta+1)$-coloring algorithm in graphs of arboricity $\alpha = \tilde{O}(1)$. Further, building on our new algorithm, we present an algorithm that improves over the longstanding time barrier (provided by Theorem 1) by a factor of $\frac{\alpha}{\Delta}$, as summarized in the following theorem.

▶ **Theorem 3.** *For any n-vertex m-edge graph of maximum degree Δ and arboricity α, there is a randomized algorithm that computes a $(\Delta + 1)$-edge coloring within time $\tilde{O}(\min\{m\sqrt{n} \cdot \frac{\alpha}{\Delta}, m\alpha\}) = \tilde{O}(\min\{m\sqrt{n}, m\Delta\}) \cdot \frac{\alpha}{\Delta}$. The time bound holds both in expectation and with high probability.*

Remark. The exact runtime bound of our algorithm (including the polylog(n) factors that are suppressed under the $\tilde{O}$-notation in Theorem 3) is the following: $O(\min\{m\sqrt{n \log n}, m\Delta \log n\}) \cdot \frac{\alpha}{\Delta} + O(m \log n)$ in expectation and $O(\min\{m\sqrt{n} \log^{1.5} n, m\Delta \log^2 n\}) \cdot \frac{\alpha}{\Delta} + O(m \log^2 n)$ with high probability. However, we made no attempt to optimize polylogarithmic factors in this work. Note that this runtime is near-linear when $\alpha = \tilde{O}(1)$ as well as when $\alpha = \tilde{O}(\frac{\Delta}{\sqrt{n}})$.

The aforementioned improvement of Sinnamon [21] to the state-of-the-art runtime bound by Gabow et al. [14] (i.e., the removal of the $\sqrt{\log n}$ factor from the term $m\sqrt{n \log n}$) was achieved via two algorithms: a rather intricate deterministic algorithm, which is very similar to a deterministic algorithm by [14], and an elegant randomized algorithm that greatly simplifies the deterministic one. Our algorithm follows closely Sinnamon's randomized algorithm, with one key difference: we *give precedence to low degree vertices and edges over high degree ones*, where the degree of an edge (which we shall refer to as its *weight*) is the minimum degree of its endpoints. Our algorithm can thus be viewed as a *degree-sensitive refinement* of Sinnamon's algorithm. The analysis of our algorithm combines several new ideas to achieve the claimed improvement in the running time.

1.1 Technical Overview and Conceptual Contribution

To compute a $(\Delta+1)$(-edge)-coloring, one can simply color the edges of the graph one after another, using what we shall refer to as a `Color-One-Edge` procedure. The most basic `Color-One-Edge` procedure is by Misra and Gries [18], and is a simplification of an algorithm by Bollobás [5]. Given a graph with some *partial* $(\Delta+1)$-coloring and an arbitrary uncolored edge $e = (u, v)$, this procedure *recolors* some edges so as to *free* a color for the uncolored edge e, and then colors e with that color. Procedure `Color-One-Edge` is carried out by (1) creating a *fan* F centered at one of e's endpoints u and *primed* by some color c_1,[2] and (2) a simple *maximal alternating path* P starting at u with edges colored by the primed color c_1 and another color c_0 that is *missing* (i.e., not occupied) on u. The runtime of Procedure `Color-One-Edge` is linear in the size of F and the length of P, hence it is $O(|F| + |P|) = O(\mathrm{d}(u) + n) = O(n)$, where $\mathrm{d}(u)$ denotes the degree of u. Applying this procedure iteratively for all edges in the graph leads to a runtime of $O(mn)$.

Instead of coloring one edge at a time via Procedure `Color-One-Edge`, Gabow et al. [14] proposed a different approach, which uses a more complex procedure, `Parallel-Color`, for coloring *multiple* uncolored edges at a time. Procedure `Parallel-Color` chooses a color c and colors, in $O(m)$ time, a constant fraction of the uncolored edges incident to vertices on which color c is missing. By applying Procedure `Parallel-Color` iteratively $O(\Delta \log n)$ times, all edges can be colored in $O(m\Delta \log n)$ time. This $O(m\Delta \log n)$-time algorithm, hereafter `Low-Degree-Color`, is fast only for graphs of small maximum degree.

Gabow et al. [14] also gave a simple recursive algorithm, hereafter `Recursive-Color-Edges`, which first partitions the edges of the graph into two separate subgraphs of maximum degree $\leq \Delta/2 + 1$, then recursively computes a $(\Delta/2 + 2)$-coloring in each subgraph via `Recursive-Color-Edges`, and it combines these two colorings into a single $(\Delta+4)$-coloring of the entire graph. Next, the algorithm uncolors all edges in the three smallest color class (of size $O(\frac{m}{\Delta})$), and finally all uncolored edges are then colored in time $O(\frac{mn}{\Delta})$ by applying the aforementioned `Color-One-Edge`. The recursion bottoms when the maximum degree is small enough ($\Delta \leq \sqrt{\frac{n}{\log n}}$), and then Algorithm `Low-Degree-Color` is applied in order to color the remaining subgraphs. As the term of $O(\frac{mn}{\Delta})$ grows geometrically with the recursion levels and as the recursion bottoms at maximum degree $O(\sqrt{\frac{n}{\log n}})$, the total runtime of `Recursive-Color-Edges` is $O(m\sqrt{n \log n})$.

[2] Such a fan is a star rooted at u that spans some of u's neighbors, with specific conditions on the colors of the edges of this star and the missing colors on their vertices. Refer to Section 2 for the definition of this and all other notions used throughout.)

Sinnamon [21] obtained a runtime of $O(m\sqrt{n})$ time. He achieved this result via two algorithms: A deterministic algorithm, which is similar to that of Gabow et al., and a much simpler and elegant randomized variant, which we briefly describe next. First, Sinnamon devised a simple random version of Procedure `Color-One-Edge`, where the only difference from the deterministic procedure is that the uncolored edge $e = (u, v)$, the endpoint u of e, and the missing color of u – are all chosen *randomly* rather than *arbitrarily*. Sinnamon observes that the runtime of this random procedure is not just $O(n)$ as before, but it is also bounded by $O(\frac{m\Delta}{l})$ *in expectation*, where l is the number of uncolored edges in the graph. Indeed, recall that the runtime of this procedure is linear in the size of the fan F and the length of the path P. Clearly, the size of a fan is bounded by the maximum degree $\Delta = O(\frac{m\Delta}{l})$. The main observation is that one can bound the expected length of the path P by $O(\frac{m\Delta}{l})$ as follows: (i) as each edge can be in at most $\Delta + 1$ such paths, the sum of lengths of all the possible maximal alternating paths is bounded by $m(\Delta + 1) = O(m\Delta)$, and (ii) the probability of the algorithm choosing any particular path is bounded by $\frac{1}{l}$.

By iteratively applying this randomized procedure `Color-One-Edge`, the expected runtime of coloring all edges is bounded by $\sum_{i=1}^{m} O(\frac{m\Delta}{m+1-i}) = O(m\Delta \sum_{i=1}^{m} \frac{1}{m+1-i}) = O(m\Delta \log n)$. This provides a much simpler randomized substitute for Algorithm `Low-Degree-Color` by [14], which Sinnamon then applied in conjunction with Algorithm `Recursive-Color-Edges` by [14] and some small tweaks to obtain an elegant randomized algorithm with expected runtime $O(m\sqrt{n})$.

1.1.1 Our approach

As mentioned, the goal of this work is to improve the longstanding runtime bound for $(\Delta + 1)$-coloring [14, 21] by a factor of $\frac{\alpha}{\Delta}$, where α is the graph's arboricity. While the required number of colors (Δ or $\Delta + 1$) grows linearly with the maximum degree Δ, the runtime might not need to grow too. The main conceptual contribution of this work is in unveiling this rather surprising phenomenon – *the running time shrinks as the maximum degree grows* (provided that the arboricity does not grow together with Δ).

In low arboricity graphs, although some vertices may have high degree (close to Δ), most vertices have low degree (close to α). Similarly, defining the degree (or *weight*) $w(e)$ of an edge e as the minimum degree of its endpoints – although some edges may have large weight, most edges have low weight (as the sum of edge weights is known to be $O(m\alpha)$). Our key insight is that giving precedence to low degree vertices and edges over high degree ones gives rise to significant improvements in the running time, for all graphs with $\alpha \ll \Delta$.

A density-sensitive `Low-Degree-Color` algorithm. Recall that the only difference between Sinnamon's `Color-One-Edge` procedure and the deterministic one by [14] is that the uncolored edge $e = (u, v)$, the endpoint u of e, and the missing color of u are all chosen *randomly* rather than *arbitrarily*. We further modify the randomized procedure of [21] by actually *making one of these random choices deterministic* – we choose u as the endpoint of e of minimum degree, so that $\mathrm{d}(u) = w(e)$. In this way the expected size of the fan computed by the algorithm is bounded by the expected weight of the sampled edge, which, in turn, is the total weight of all uncolored edges divided by their number, yielding an upper bound of $O(\frac{m\alpha}{l})$ – which refines the aforementioned bound of $O(\frac{m\Delta}{l})$ by [21]. Next, we would like to argue that the expected length of a maximal alternating path P is $O(\frac{m\alpha}{l})$. While the argument for bounding the probability that the algorithm chooses any particular path is $O(\frac{1}{l})$ remains pretty much the same, the challenge is in bounding the sum of lengths of all maximal alternating paths by $O(m\alpha)$ rather than $O(m\Delta)$. Our key insight here is to bound the total lengths of the *internal* parts of the paths, by exploiting the observation that *any edge e can be internal in at most $w(e)$ maximal alternating paths*, which directly implies that the sum of lengths is bounded by the sum of the weights of the colored edges, which, in turn is bounded by $O(m\alpha)$. As a result, we improve the runtime of Procedure `Low-Degree-Color` from $O(m\Delta\log n)$ to $O(m\alpha\log n) = O(m\Delta\log n)\cdot\frac{\alpha}{\Delta}$. Refer to Section 3 for the full details.

A density-sensitive `Recursive-Color-Edges` algorithm. The aforementioned improvement provided by our density-sensitive `Low-Degree-Color` algorithm only concerns the regime of low Δ. To shave off a factor of $\frac{\alpha}{\Delta}$ in the entire regime of Δ, we need to come up with a density-sensitive `Recursive-Color-Edges` algorithm – which is the main technical challenge of this work.

In the original recursive algorithm of Gabow et al. [14], the starting point is that any graph can be partitioned in linear time into two subgraphs in which the degrees of all vertices, including the maximum degree, reduce by a factor of 2. To achieve a density-sensitive refinement of Algorithm `Recursive-Color-Edges`, the analog starting point would be to partition the graph into two subgraphs of half the arboricity. Although this guarantee can be achieved via a random partition, we overcome this issue deterministically, by *transitioning from the graph's arboricity to the graph's normalized weight*, where the *normalized weight* of the graph is defined as the ratio of the graph's weight (the sum of its edge weights) to the number of edges. Indeed, since the degrees of vertices decay by a factor of 2 at each recursion level, so does the weight of each subgraph. An added benefit of this approach is that all our results extend from graphs of arboricity α to graphs of normalized weight 2α, which forms a much wider graph class.[3]

In the original recursive algorithm of [14], to transition from a $(\Delta+4)$-coloring to a $(\Delta+1)$-coloring, the algorithm uncolors all edges in the three color classes of smallest size, and then colors all those $O(\frac{m}{\Delta})$ edges by applying the `Color-One-Edge` procedure, which takes time $O(\frac{mn}{\Delta})$. To outperform the original algorithm, our algorithm uncolors all edges in the three color classes of smallest *weight* (rather than size), implying that the total weight of those uncolored edges is bounded by $O(\frac{m\alpha}{\Delta})$. As a result, we can no longer argue that the total number of those edges is $O(\frac{m}{\Delta})$ as before, but we demonstrate that the new upper bound on the weights gives rise to a faster algorithm.

We improve the bound for coloring the uncolored edges at every level of recursion, from $O(\frac{mn}{\Delta})$ to $O(\frac{mn}{\Delta})\cdot\frac{\alpha}{\Delta}$, as follows. First, consider the time spent on uncolored edges of *high weights*, i.e., edges of weight $> \Delta/2$. Since the number of such edges is bounded by $O(\frac{m\alpha}{\Delta})/(\Delta/2) = O(\frac{m}{\Delta})\cdot\frac{\alpha}{\Delta}$, the simple $O(n)$ time bound of `Color-One-Edge` by [18] yields the required total runtime bound of $O(\frac{mn}{\Delta})\cdot\frac{\alpha}{\Delta}$.

The challenging part is to bound the time spent on uncolored edges of *low weights*, i.e., edges of weight $\leq \Delta/2$. Towards this end, we employ a stronger version of the bound on the expected runtime of `Color-One-Edge`. Instead of using the $O(\frac{m\alpha}{l})$ bound mentioned before, we build on the fact that any chosen endpoint of a low weight edge has $\Omega(\Delta)$ possible missing colors to choose from, which reduces the probability to choose any alternating path by a factor of Δ. With a bit more work, we can show that the expected time for coloring a low weight uncolored edge is bounded by $O(\frac{m\alpha}{l\cdot\Delta})$, yielding a total runtime bound of at most $O(\sum_{i=1}^{m}\frac{m\alpha}{(m+1-i)\cdot\Delta}) = O(\frac{m\alpha}{\Delta}\log n)$ for all the low weight uncolored edges. Although this upper bound is better than the required one by factor of $\frac{n}{\Delta}$, the bottleneck stems from the edges of high weights.

[3] Any graph of arboricity α has normalized weight at most 2α (refer to Claim 4), but a graph of normalizd weight α may have arboricity much larger than α.

As we demonstrate in Section 4, by combining these ideas with a careful analysis, our randomized `Recursive-Color-Edges` Algorithm achieves the required runtime bound of $O(m\sqrt{n}\log n)\cdot\frac{\alpha}{\Delta}$.

1.2 Related Work

Edge coloring is a fundamental graph problem that has been studied over the years in various settings and computational models; we shall aim for brevity. Surprisingly perhaps, the body of work on fast sequential $(\Delta+1)$-coloring algorithms, which is the focus of this work, is quite sparse. We have already discussed in detail the $(\Delta+1)$-coloring algorithms of [14, 18, 21]; a similar result to Gabow et al. [14] was obtained independently (and earlier) by [1]. It is NP-hard to determine whether a graph can be colored with Δ colors or not [16]. On the other hand, for restricted graph families, particularly biparite graphs and planar graphs of sufficiently large degree, near-linear time algorithms are known [13, 11],[10]. Recently, $(\Delta+1)$-coloring algorithms across different models of computation were given in [9].

There are also known edge coloring algorithms for bounded arboricity graphs, but they are slow or they use many colors. In graphs of *constant* arboricity α, a near-linear time $(\lceil\frac{(\alpha+2)^2}{2}\rceil-1)$-coloring algorithm was given in [25]. For graphs whose arboricity α is smaller than Δ by at least a constant factor, several Δ-coloring algorithms are known [12, 15, 20, 23, 24, 6], but their running time is at least $O(mn)$.

1.3 Subsequent Work

Kowalik [17] gave a Δ-edge coloring algorithm with a running time of $\tilde{O}(m\alpha^3)$, for graphs whose arboricity α is smaller than Δ by at least a constant factor.[4] The work of [17] crucially builds on top of our techniques; indeed, without relying on our techniques, the running time of the algorithm of [17] would grow to $\tilde{O}(m\Delta^3)$, which is $\tilde{O}(mn^3)$ in bounded arboricity graphs of large degree; this bound, in turn, is slower by a factor of n^2 than some of the aforementioned Δ-edge coloring algorithms in bounded arboricity graphs [12, 15, 20, 23, 24, 6].

For a dynamically changing graph with maximum degree Δ and arboricity α, [8] and [4] independently showed how to maintain a $(\Delta+O(\alpha))$-coloring in $\tilde{O}(1)$ update time.

Very recently, [3] and [2] independently obtained faster algorithms for $(\Delta+1)$-edge coloring in general graphs, obtaining running times of $\tilde{O}(mn^{1/3})$ and $\tilde{O}(n^2)$ respectively.

2 Preliminaries

We work in the standard word RAM model of computation, with words of size $w:=\Theta(\log n)$. In particular, we can index any of the $2^{O(w)}=\text{poly}(n)$ memory addresses, perform basic arithmetic on w-bit words, and sample w-bit random variables, all in constant time. In App. B of the full version of our paper, we describe the data structures used by our algorithms; they are all basic and easy to implement, and their space usage is linear in the graph size.

We denote the degree of a vertex v by $\mathrm{d}(v)$. The *weight* $w(e)$ of an edge $e=(u,v)$ is defined as the minimum degree of its endpoints, i.e., $w(e):=\min\{\mathrm{d}(u),\mathrm{d}(v)\}$. The *weight* $w(G)$ of a graph $G=(V,E)$ is defined as the sum of weights over its edges, i.e., $w(G):=\sum_{e\in E}w(e)$.

Note that the weight $w(G)$ of any m-edge graph G satisfies $w(G)\geq m$. The following claim, due to [7], asserts that the weight of any m-edge graph exceeds m by at most a factor of 2α.

[4] More precisely, the result holds for graphs whose *maximum average degree* is at most $\Delta/2$; the maximum average degree is within a constant factor from the graph's arboricity.

▷ Claim 4 (Lemma 2 in [7]). For any m-edge graph G with arboricity α, we have $w(G) \leq 2m\alpha$.

In what follows, let $G = (V, E)$ be an arbitrary n-vertex m-edge graph, and we let Δ to the maximum degree of G. For any integer $k \geq 1$, let $[k]$ denote the set $\{1, 2, ..., k\}$.

▶ **Definition 5.** *A* (proper) partial k-(edge-)coloring χ *of* G *is a color function* $\chi : E \rightarrow [k] \cup \{\neg\}$ *such that any two distinct colored edges* e_1, e_2 *that share an endpoint do not receive the same color. An edge* e *with* $\chi(e) \in [k]$ *is said to be* colored *(by* χ*), whereas an edge* e *with* $\chi(e) = \neg$ *is said to be* uncolored*. If all the edges in* G *are colored by* χ*, we say that* χ *is a* (proper) k-(edge-)coloring.

Given a partial k-coloring χ for G, we define $M(v)$ as the set of *missing* colors of v, i.e., the set of colors in the color palette $[k]$ not occupied by any of the incident edges of v. For a partial $(\Delta + 1)$-coloring, $M(v)$ is always nonempty, as v has at most Δ neighbours and there are $\Delta + 1$ colors to choose from.

▶ **Definition 6** (Colored and uncolored edges). *Consider a partial* k*-coloring* χ*, where all the edges in* G *are colored but* l*. We define* $E_{\text{cl}}(G, \chi) = E_{\text{cl}}$ *as the set of* $m - l$ *colored edges of* G *and* $E_{\text{un}}(G, \chi) = E_{\text{un}}$ *as the set of* l *uncolored edges of* G*. If all edges of* G *are uncolored, i.e.,* $l = m$*, we say that* χ *is an* empty *coloring.*

2.1 Fans

In what follows we let χ be a proper partial $(\Delta + 1)$-coloring of G.

▶ **Definition 7.** *A* fan F *is a sequence of vertices* $(v, x_0, ..., x_t)$ *such that* $x_0, ..., x_t$ *are distinct neighbors of* v*, the edge* (v, x_0) *is uncolored, the edge* (v, x_i) *is colored for every* $i \in [t]$*, and the color* $\chi(v, x_i)$ *is missing at vertex* x_{i-1} *for every* $i \in [t]$*. The vertex* v *is called the* center *of* F*, and* $x_0, ..., x_t$ *are called the* leaves *of* F*.*

The useful property of a fan is that *rotating* or *shifting* the colors of the fan preserves the validity of the coloring. Let $F = (v, x_0, ..., x_t)$ be a fan and write $c_i = \chi(v, x_i)$, for each $i = 1, ..., t$. To *rotate* or *shift* F from x_j means to set $\chi(v, x_{i-1}) = c_i$ for every $i = 1, ..., j$ and make (v, x_j) uncolored. After the rotation or shift, the function χ is still a proper partial coloring, but now (v, x_j) is uncolored instead of (v, x_0). Note that $M(v)$ is unaffected by the shift. Refer to App. A in the full version of our paper for illustrations of fans and fan shifting.

To extend a partial $(\Delta + 1)$-coloring (i.e., increase the number of colored edges), we shall focus on *maximal* fans (which cannot be further extended), using the following definition.

▶ **Definition 8.** *A fan* $F = (v, x_0, ..., x_t)$ *is said to be* primed *by color* $c_1 \in M(x_t)$ *if one of the following two conditions hold: (1)* $c_1 \in M(v)$*, or (2)* $c_1 \in M(x_j)$ *for some* $j < t$*.*

Computing a primed fan. Given an uncolored edge (v, x_0), we can get a primed fan $(v, x_0, ..., x_t)$ via Algorithm 1 (refer to [14, 18, 21]):

▶ **Lemma 9.** *Algorithm* `Make-Primed-Fan` *returns a primed fan with center* v *in* $O(\mathrm{d}(v))$ *time.*

Proof. By the description of the algorithm, it is immediate that `Make-Primed-Fan` returns a primed fan. The number of iterations of the while loop is bounded by $\mathrm{d}(v)$, since every iteration in which the loop does not terminate adds a new neighbor of v as a leaf of F. To complete the proof, we argue that each iteration can be implemented in constant time. We can store the at most $\mathrm{d}(v)$ leaves that are added to F in a hash table, so that line 9 can be implemented in constant time. The remaining part of an iteration can be carried out in constant time in the obvious way using the data structures mentioned in App. B if the full version of our paper. ◀

Algorithm 1 `Make-Primed-Fan`$(G, \chi, (v, x_0))$.

Input: A graph G with a partial $(\Delta + 1)$-coloring χ and an uncolored edge (v, x_0)
Output: A fan $F = (v, x_0, ..., x_t)$ and color $c_1 \in M(x_t)$ such that F is primed by c_1

1 $F \leftarrow (v, x_0)$
2 $t \leftarrow 0$
3 **while** F *is not primed* **do**
4   Pick any $c_1 \in M(x_t)$
5   **if** $c_1 \in M(v)$ **then**
6   **return** F, c_1
7   **else**
8   Find the edge $e = (v, x_{t+1})$ such that $\chi(e) = c_1$
9   **if** $x_{t+1} \in x_1, ..., x_t$ **then**
10   **return** F, c_1
11   **else**
12   Append x_{t+1} to F
13             $t \leftarrow t + 1$

2.2 Alternating Paths

▶ **Definition 10.** *We say that path P is a (c_0, c_1)-*alternating path*, for a pair c_0, c_1 of distinct colors, if P consists of edges with colors c_0 and c_1 only. We say that a (c_0, c_1)-alternating path P is* maximal *if $P = (v_0, v_1, \dots, v_{|P|})$ and both v_0 and $v_{|P|}$ have only one edge colored by c_0 and c_1 (hence P cannot be extended further). Although a maximal alternating path may form a cycle, we shall focus on simple paths.*

The useful property of a maximal alternating path is that *flipping* the colors of the path edges preserves the validity of the coloring. That is, let $P = e_1 \circ \dots \circ e_{|P|}$ be a maximal (c_0, c_1)-alternating path such that for every $i = 1, \dots, |P|$: if $i \equiv 1 \pmod 2$ then $\chi(e_i) = c_0$ and if $i \equiv 0 \pmod 2$ then $\chi(e_i) = c_1$. *Flipping* P means to set the function χ, such that for every $i = 1, ..., |P|$: if $i \equiv 1 \pmod 2$ then $\chi(e_i) = c_1$ and if $i \equiv 0 \pmod 2$ then $\chi(e_i) = c_0$; the resulting function χ after the flip operation is a proper partial edge-coloring.

Flipping a maximal (c_0, c_1)-alternating path that starts at a vertex v is a useful operation, as it "frees" for v the color c_0 that is occupied by it, replacing it with the missing color c_1 on v. Refer to App. A in the full version of our paper for an illustration of maximal alternating path flipping.

2.3 Algorithm `Extend-Coloring`: Naively Extending a Coloring by One Edge

Given an uncolored edge e, there is a standard way to color it and by that to extend the coloring, using a primed fan F rooted at one of the endpoints of e, say v, as well as a maximal alternating path P starting at v, by flipping the path and then rotating a suitable prefix of the fan, as described in the following algorithm (refer to [14, 18, 21]):

Algorithm 2 `Extend-Coloring`(G, χ, e, F, c_1, P).

Input: A graph G with a partial $(\Delta+1)$-coloring χ, an uncolored edge e=(v, x_0), a fan $F = (v, x_0, ..., x_t)$ primed by color c_1 and a maximal (c_0, c_1)-alternating path P starting at v

Output: Updated partial $(\Delta+1)$-coloring χ of G, such that e is also colored by χ

if $c_1 \in M(v)$ **then**
    Shift F from x_t
    $\chi(v, x_t) \leftarrow c_1$
else
    $x_j \leftarrow$ leaf of F with $\chi(v, x_j) = c_1$ (when $j < t$)
    $w \leftarrow$ other endpoint of P (other than v)
    Flip P
    if $w \neq x_{j-1}$ **then**
        Shift F from x_{j-1}
        $\chi(v, x_{j-1}) \leftarrow c_1$
    else
        Shift F from x_t
        $\chi(v, x_t) \leftarrow c_1$
return χ

▶ **Lemma 11.** *Algorithm* `Extend-Coloring` *(Algorithm 2), when given as input a proper partial* $(\Delta+1)$*-coloring* χ*, the first uncolored edge* (v, x_0) *of a fan* F *primed by a color* c_1 *and a maximal* (c_0, c_1)*-alternating path* P*, where* c_0 *is free on* v*, extends the coloring* χ *into a proper partial* $(\Delta+1)$*-coloring, such that* e*, as well as all previously colored edges by* χ*, is also colored by* χ*. Moreover, this algorithm takes* $O(|F| + |P|) = O(\mathrm{d}(v) + |P|)$ *time.*

Proof. This algorithm provides the standard way to extend a coloring by one edge. (We omit the correctness proof for brevity; refer to [14, 18, 21] for the proof.)

As for the running time, there are three different cases to consider: (1) $c_1 \in M(v)$, (2) $c_1 \notin M(v)$ and $w \neq x_{j-1}$, (3) $c_1 \notin M(v)$ and $w = x_{j-1}$. Using our data structures (described in App. B in the full version of our paper), one can identify the case and find the leaf x_j and the endpoint w (if needed) in time $O(|F| + |P|)$. Beyond that, in each case the algorithm performs at most one path flip, one fan shift, and one edge coloring, which also takes time $O(|F| + |P|)$. As F contains at most $\mathrm{d}(u)$ leaves, the total running time is $O(|F| + |P|) = O(\mathrm{d}(v) + |P|)$. ◀

3 A $(\Delta+1)$-Coloring Algorithm with Runtime $\tilde{O}(m\alpha)$

3.1 Internal Edges of Maximal Alternating Paths

The paths that we shall consider are maximal alternating paths that start and finish at different vertices (i.e., we do not consider cycles). A vertex v in a path P is called *internal* if it is not one of the two endpoints of P. An edge $e = (u, v)$ of path P is called *internal* if both u and v are internal vertices of P. For a path P, denote by $I(P)$ the set of internal edges of P. We shall use the following immediate observation later.

▶ **Observation 12.** *For any path* P*,* $|P| \leq |I(P)| + 2$*.*

Any edge may serve as an internal edge in possibly many different maximal alternating paths. The following key lemma bounds the number of such paths by the edge's weight.

▶ **Lemma 13.** *Any colored edge e can be an internal edge of at most $w(e)$ maximal alternating paths.*

Proof. Let $e = (u, v)$ be a colored edge with color c_e. For any color $c_2 \neq c_e$, we note that e can be internal in a maximal (c_e, c_2)-alternating path P only if each among u and v is incident on an edge with color c_2, thus the number of such colors c_2 is bounded by $\min\{\mathrm{d}(u), \mathrm{d}(v)\} = w(e)$. To complete the proof, we note that there is at most one maximal (c_e, c_2)-alternating path that contains e, for any color c_2. ◀

For a graph G with a given coloring χ, denote by $MP = MP(G, \chi)$ the set of maximal alternating paths in G induced by χ.

▶ **Lemma 14.** $\sum_{P \in MP} |I(P)| \leq \sum_{e \in E_{\mathrm{cl}}} w(e)$.

Proof.

$$\sum_{P \in MP} |I(P)| \;=\; \sum_{P \in MP} \sum_{e \in I(P)} 1 \;=\; \sum_{e \in E} \sum_{P \in MP : e \in I(P)} 1 \;=\; \sum_{e \in E_{\mathrm{cl}}} \sum_{P \in MP : e \in I(P)} 1 \;\leq\; \sum_{e \in E_{\mathrm{cl}}} w(e),$$

where the last inequality follows from Lemma 13, as $\sum_{P \in MP : e \in I(P)} 1$ is just the number of maximal alternating paths in which the colored edge e is an internal edge. ◀

3.2 Algorithm `Color-One-Edge`: An Algorithm for Coloring a Single Edge

The following algorithm is based on Algorithm `Random-Color-One` of [21], but with one crucial tweak: Given the chosen random edge, we focus on the endpoint of the edge of *minimum degree*.

Algorithm 3 `Color-One-Edge`(G, χ).

Input: A graph G with a partial $(\Delta + 1)$-coloring χ
Output: Updated partial $(\Delta + 1)$-coloring χ of G, such that one more edge is colored by χ

1 $e = (u, v) \leftarrow$ A random uncolored edge
2 Assume w.l.o.g. that $\mathrm{d}(u) \leq \mathrm{d}(v)$
3 $F, c_1 \leftarrow$ `Make-Primed-Fan`$(G, \chi, (u, v))$
4 Choose a random color $c_0 \in M(u)$
5 $P \leftarrow$ The maximal (c_0, c_1)-alternating path starting at u
6 $\chi \leftarrow$`Extend-Coloring`(G, χ, e, F, c_1, P)
7 **return** χ

▶ **Lemma 15.** *Let G be a graph of maximum degree Δ and let χ be a partial $(\Delta+1)$-coloring of all the edges in G but l. Then the expected runtime of `Color-One-Edge` on G and χ is $O(\frac{w(G)}{l}) = O(\frac{m\alpha}{l})$.*

Proof. Denote the runtime of Algorithm `Color-One-Edge` = `Color-One-Edge`(G, χ) by T(`Color-One-Edge`).

Let $e_r = (u_r, v_r)$ be the random uncolored edge chosen in line 1 of the algorithm, with $\mathrm{d}(u_r) \leq \mathrm{d}(v_r)$, let c_r be the random missing color chosen in line 4 of the algorithm, and let

P_r be the maximal alternating path starting at u_r obtained in line 5 of the algorithm. By definition, both $|P_r|$ and $\mathrm{d}(u_r)$ are random variables.

We will prove the lemma as a corollary of the following three claims:

▷ Claim 16. $\mathbb{E}[T(\texttt{Color-One-Edge})] = O(\mathbb{E}[|P_r|] + \mathbb{E}[\mathrm{d}(u_r)])$.

▷ Claim 17. $\mathbb{E}[|P_r|] = O(1 + \frac{1}{l}\sum_{e\in E_{\text{cl}}} w(e))$.

▷ Claim 18. $\mathbb{E}[\mathrm{d}(u_r)] = O(\frac{1}{l}\sum_{e\in E_{\text{un}}} w(e))$.

Proof of Claim 16. We can pick a random uncolored edge in constant time. In addition, we can pick a random missing color on a vertex in (expected) time linear to its degree. Also, we can compute the path P_r in $O(|P_r|)$ time by repeatedly adding edges to it while possible to do. Consequently, by Lemmas 9 and 11, which bound the running time of Algorithms `Make-Primed-Fan` and `Extend-Coloring` by $O(|P_r|) + O(\mathrm{d}(u_r))$, we conclude that $\mathbb{E}[T(\texttt{Color-One-Edge})] = O(\mathbb{E}[|P_r|] + \mathbb{E}[\mathrm{d}(u_r)])$. ◁

Proof of Claim 17. By Observation 12 $\mathbb{E}[|P_r|] \le \mathbb{E}[|I(P_r)|] + O(1)$. We next prove that $\mathbb{E}[|I(P_r)|] = O(\frac{1}{l}\sum_{e\in E_{\text{cl}}} w(e))$.

For every maximal alternating path P in MP, let $u_0(P)$ and $u_{|P|}(P)$ be the first and last endpoints of P, respectively, and let $c_0(P)$ and $c_{|P|}(P)$ be the missing colors of $u_0(P)$ and $u_{|P|}(P)$ from the two colors of P, respectively. For a vertex v, denote by $l(v)$ the number of uncolored edges incident on v. Note that for every P in MP,

$$\begin{aligned}
\mathbb{P}(P_r = P) &\le \mathbb{P}(c_r = c_0(P) \mid u_r = u_0(P)) \cdot \mathbb{P}(u_r = u_0(P)) \\
&\quad + \mathbb{P}(c_r = c_{|P|}(P) \mid u_r = u_{|P|}(P)) \cdot \mathbb{P}(u_r = u_{|P|}(P)) \\
&\le \frac{1}{|M(u_0(P))|} \cdot \frac{l(u_0(P))}{l} + \frac{1}{|M(u_{|P|}(P))|} \cdot \frac{l(u_{|P|}(P))}{l} \\
&\le \frac{1}{|M(u_0(P))|} \cdot \frac{|M(u_0(P))|}{l} + \frac{1}{|M(u_{|P|}(P))|} \cdot \frac{|M(u_{|P|}(P))|}{l} = \frac{2}{l}.
\end{aligned}$$

It follows that

$$\mathbb{E}[|I(P_r)|] = \sum_{P\in MP} \mathbb{P}(P_r = P)\cdot|I(P)| \le \sum_{P\in MP} \frac{2}{l}\cdot|I(P)| \overset{\text{(Lemma 14)}}{\le} \sum_{e\in E_{\text{cl}}} \frac{2}{l}\cdot w(e)$$

which is $O\left(\frac{1}{l}\sum_{e\in E_{\text{cl}}} w(e)\right)$. ◁

Proof of Claim 18. Define $D_m(u)$ to be the set of uncolored edges $e = (u, v)$, such that $\mathrm{d}(u) \le \mathrm{d}(v)$. Now,

$$\mathbb{P}(u_r = u) \le \mathbb{P}\left(e_r = (u, v) \cap \mathrm{d}(u) \le \mathrm{d}(v)\right) = \frac{|D_m(u)|}{l}.$$

Consequently,

$$\begin{aligned}
\mathbb{E}[\mathrm{d}(u_r)] &= \sum_{u\in V} \mathbb{P}(u_r = u)\cdot \mathrm{d}(u) \le \sum_{u\in V} \frac{|D_m(u)|}{l}\cdot \mathrm{d}(u) = \frac{1}{l}\sum_{u\in V}\sum_{e\in D_m(u)} \mathrm{d}(u) \\
&= \frac{1}{l}\sum_{e\in E_{\text{un}}}\sum_{x: e\in D_m(x)} \mathrm{d}(x) \le \frac{1}{l}\sum_{e=(u,v)\in E_{\text{un}}} 2\cdot\min\{\mathrm{d}(u), \mathrm{d}(v)\}
\end{aligned}$$

which is $O\left(\frac{1}{l}\sum_{e\in E_{\text{un}}} w(e)\right)$. ◁

Now we are ready to complete the proof of Lemma 15. Using Claims 16, 17 and 18 we get

$$\mathbb{E}[T(\texttt{Color-One-Edge})] \;=\; O(\mathbb{E}[|P_r|] + \mathbb{E}[\mathrm{d}(u_r)]) \;=\; O\left(1 + \frac{1}{l}\sum_{e \in E_{\mathrm{cl}}} w(e) + \frac{1}{l}\sum_{e \in E_{\mathrm{un}}} w(e)\right)$$

which is $O\left(\frac{w(G)}{l}\right)$. Recalling that $w(G) = O(m\alpha)$ holds by Claim 4 completes the proof. ◀

3.3 Algorithm `Color-Edges`: Coloring a Single Edge Iteratively

The following algorithm is given as input a graph G and a partial $(\Delta + 1)$-coloring χ, and it returns as output a proper $(\Delta + 1)$-coloring for G. This algorithm is identical to the analogous one by [21]; however, here we invoke our modified `Color-One-Edge` Algorithm in line 2 rather than the analogous one by [21].

Algorithm 4 `Color-Edges`(G, χ).

Input: A graph G with a partial $(\Delta + 1)$-coloring χ
Output: Updated partial $(\Delta + 1)$-coloring χ of G, such that all the edges in G are colored

1 **while** $E_{\mathrm{un}} \neq \emptyset$ **do**
2 │ $\chi \leftarrow$ `Color-One-Edge`(G, χ)
3 **return** χ

▶ **Lemma 19.** *The expected runtime of Algorithm `Color-Edges` on a graph G with an empty partial $(\Delta + 1)$-coloring χ is $O(w(G)\log n) = O(m\alpha \log n)$.*

Proof. As described in App. B of the full version of our paper, we can check whether $E_{\mathrm{un}} = \emptyset$ in constant time. Since each call to Algorithm `Color-One-Edge` colors a single uncolored edge, the while loop consists of m iterations. Also, the runtime of each iteration is that of the respective call to `Color-One-Edge`. At the beginning of the ith iteration, the number of uncolored edges l is $m - i + 1$, so by Lemma 15, the expected runtime of the ith iteration, denoted by $\mathbb{E}[T(i\text{th iteration})]$, is $O(\frac{w(G)}{m-i+1})$. It follows that

$$\mathbb{E}[T(\texttt{Color-Edges})] \;=\; \sum_{i=1}^{m} \mathbb{E}[T(i\text{th iteration})] \;=\; O\left(\sum_{i=1}^{m} \frac{w(G)}{m - i + 1}\right) \;=\; O(w(G)\log n).$$

Recalling that $w(G) = O(m\alpha)$ holds by Claim 4 completes the proof. ◀

4 Our $(\Delta + 1)$-Coloring Algorithm: `Recursive-Color-Edges`

In this section we present our $(\Delta + 1)$-edge-coloring algorithm, `Recursive-Color-Edges`, which proves Theorem 3. Our algorithm is similar to that of [21], which, in turn, in based on Gabow et al. [14] – except for a few small yet crucial tweaks, one of which is that we employ our algorithm `Color-Edges` as a subroutine rather than the analogous subroutine from [21]. We first describe the approach taken by [14, 21], and then present our algorithm and emphasize the specific modifications needed for achieving the improvement in the running time.

4.1 The Approach of [14, 21]

The algorithm of [14, 21] employs a natural divide-and-conquer approach. It partitions the input graph into two edge-disjoint subgraphs of maximum degree roughly $\Delta/2$, then it recursively computes a coloring with at most $\Delta/2+2$ colors for each subgraph separately, and then it stitches together the two colorings into a single coloring. Naively stitching the two colorings into one would result in up to $\Delta+4$ colors, so the idea is to prune excessive colors and then deal with the remaining uncolored edges via a separate coloring algorithm.

In more detail, the algorithm consists of four phases: **Partition**, **Recurse**, **Prune**, and **Repair**.

- **Partition.** The algorithm partitions the edges of the graph into two edge-disjoint subgraphs, such that the edges incident on each vertex are divided between the two subgraphs almost uniformly. This in particular implies that the maximum degree in each subgraph is roughly $\Delta/2$. Such a partition can be achieved by a standard procedure, *Euler partition*, which was used also by [14, 21]. For completeness, in Section 4.2 we describe this procedure and prove some basic properties that will be used later.
- **Recurse.** The algorithm recursively computes a coloring with at most $\Delta/2+2$ colors for each subgraph separately, where the two colorings use disjoint palettes of colors. Then, we combine the two colorings into one by simply taking their union, which results with a proper coloring with at most $\Delta+4$ colors.
- **Prune.** At this point, the number of colors used in the coloring is Δ', for $\Delta' \leq \Delta+4$, which exceeds the required bound of $\Delta+1$. To prune the up to three extra colors, the algorithm groups the edges into color classes, chooses the $\Delta'-(\Delta+1)$ color classes of smallest size, and then uncolors all edges in those color classes. *In our algorithm, we choose the $\Delta'-(\Delta+1)$ color classes of smallest* weight*, where the weight of a color class is the sum of weights of edges with that color.*
- **Repair.** To complete the partial coloring into a proper $(\Delta+1)$-coloring, each of the uncolored edges resulting from the **Prune** phase has to be recolored; this is done by a separate coloring algorihtm. *In our algorithm, this separate coloring algorithm is Algorithm `Color-Edges` from Section 3.*

4.2 The `Euler Partition` Procedure

An *`Euler partition`* is a partition of the edges of a graph into a set of edge-disjoint tours, such that every odd-degree vertex is the endpoint of exactly one tour, and no even-degree vertex is an endpoint of any tour (some tours may be cycles). Such a partition can be computed greedily in linear time, by simply removing maximal tours of the graph until no edges remain. The edges of the graph can then be split between the subgraphs by traversing each tour and alternately assigning the edges to the two subgraphs. Internal edges of the tours split evenly between the two subgraphs, so only two cases may cause an unbalanced partition:

- **Case 1: The two endpoints of any tour.** The only edge incident on any tour endpoint is assigned to one of the two subgraphs, which causes at most 1 extra edge per vertex in one of the two subgraphs.
- **Case 2: A single vertex in any tour that is an odd length cycle.** The cycle edges can be split evenly among all vertices of the cycle but one, the *starting vertex*, for which there are 2 extra edges in one of the two subgraphs. Note that the algorithm is free to choose (1) any of the cycle vertices as the starting vertex, and (2) the subgraph among the two to which the 2 extra edges would belong; the algorithm will carry out these choices so as to minimize the discrepancy in degrees of any vertex.

The next observation follows directly from the description of the `Euler partition` procedure (and by handling the two aforementioned cases that may cause an unbalanced partition in the obvious way).

▶ **Observation 20.** *Let G be a graph and let G_1, G_2 be the subgraphs of G obtained by the `Euler partition` procedure. Then for every vertex $v \in G$, the degrees of v in G_1 and G_2, denoted by $\mathrm{d}_{G_1}(v)$ and $\mathrm{d}_{G_2}(v)$ respectively, satisfy*

$$\frac{1}{2}\mathrm{d}_G(v) - 1 \ \leq \ \mathrm{d}_{G_1}(v), \mathrm{d}_{G_2}(v) \ \leq \ \frac{1}{2}\mathrm{d}_G(v) + 1.$$

4.2.1 Analysis of the `Euler Partition` procedure

Let G be an n-vertex m-edge graph, and consider any algorithm A that applies the `Euler Partition` procedure recursively. That is, upon recieving the graph G as input, Algorithm A splits G into two subgraphs G_1 and G_2 *using the `Euler Partition` procedure*, and then recursively applies Algorithm A on G_1 and G_2.

Consider the binary recursion tree of algorithm A: The first *level* L_0 consists of the root node that corresponds to the graph G. The next *level* L_1 consists of two nodes that correspond to the two subgraphs of G obtained by applying the `Euler Partition` procedure on G. In general, the *ith level* L_i consists of 2^i nodes that correspond to the 2^i subgraphs of G obtained by applying the `Euler Partition` procedure on the 2^{i-1} subgraphs of G at level L_{i-1}. In App. C of the full version of our paper, we prove some basic properties of the recursion tree of Algorithm A.

4.3 The Algorithm

In this section we present our $(\Delta+1)$-coloring algorithm. As mentioned, our algorithm follows closely that of [14, 21], but we introduce the following changes: (1) In the **Prune** phase, we uncolor edges from the three color classes of minimum weight (rather than minimum size); (2) in the **Repair** phase, as well as at the bottom level of the recursion, we invoke our modified `Color-Edges` algorithm rather than the analogous one by [21, 14]. `Recursive-Color-Edges` (Algorithm 5) gives the pseudocode for our algorithm.

4.4 Analysis of The Algorithm

We consider the binary recursion tree of Algorithm `Recursive-Color-Edges`, and note that this algorithm can assume the role of Algorithm A in Section 4.2.1; in particular, we follow the notation of Section 4.2.1.

▶ **Lemma 21.** *For any subgraph $H = (V_H, E_H)$ of G at level L_i with m_H edges, maximum degree Δ_H and weight W_H, the expected runtime spent on H due to the call to `Recursive-Color-Edges` on G is bounded by*

$$O\left(m_H + \frac{W_H}{\Delta_H} \cdot \left(\frac{n}{\Delta_H} + \log n\right)\right).$$

Proof. Consider first the case $\Delta_H \leq 2\sqrt{\frac{n}{\log n}}$. The time of the creation of the empty coloring (in line 2 of the code) is $O(m_H)$. By Lemma 19, the expected time spent on H while running `Color-Edges` (in line 3) is bounded by

$$O(W_H \log n) \ = \ O\left(m_H + W_H \frac{n}{\Delta_H^2}\right) \qquad\qquad = \ O\left(m_H + \frac{W_H}{\Delta_H} \cdot \frac{n}{\Delta_H}\right)$$

Algorithm 5 `Recursive-Color-Edges`(G).

Input: A graph G
Output: A $(\Delta+1)$-coloring χ of G
if $\Delta \leq 2\sqrt{\frac{n}{\log n}}$ **then**
  $\chi \leftarrow$ Empty $(\Delta+1)$-Coloring of G
  $\chi \leftarrow$ `Color-Edges`(G, χ)
  return χ
Partition:
Decompose G into subgraphs G_1 and G_2 by applying `Euler partition`
Recurse: // color G_1 and G_2 by at most $\Delta+4$ colors
$\chi_1 \leftarrow$ Empty $(\Delta_{G_1}+1)$-coloring of G_1 // $\Delta_{G_1} = \Delta(G_1)$
$\chi_2 \leftarrow$ Empty $(\Delta_{G_2}+1)$-coloring of G_2 // $\Delta_{G_2} = \Delta(G_2)$
$\chi_1 \leftarrow$ `Recursive-Color-Edges`(G_1, χ_1)
$\chi_2 \leftarrow$ `Recursive-Color-Edges`(G_2, χ_2)
$\chi \leftarrow (\Delta+4)$-coloring of G obtained by merging χ_1 and χ_2
Prune:
while *there are more than* $\Delta+1$ *colors in* χ **do**
  $c_l \leftarrow$ a color that minimizes $w(c_l) := \sum_{e\in E: e \text{ colored by } c_l} w(e)$
  Uncolor all edges colored by c_l // The color c_l is removed from χ
Repair: // color all uncolored edges
$\chi \leftarrow$ `Color-Edges`(G, χ)
return χ

$$= O\left(m_H + \frac{W_H}{\Delta_H} \cdot \left(\frac{n}{\Delta_H} + \log n\right)\right).$$

We may henceforth assume that $\Delta_H > 2\sqrt{\frac{n}{\log n}}$. Next, we analyze the time required by every phase of the algorithm. We first note that the first three phases of the algorithm can be implemented in $O(m_H)$ time:

- **Partition:** The `Euler partition` procedure takes $O(m_H)$ time, as explained in Section 4.2.
- **Recurse:** We only consider the time spent at the **Recurse** phase on H itself, i.e., the time needed to create the two empty colorings χ_1 and χ_2 and the time needed to merge them into χ, each of which takes time $O(m_H)$.
- **Prune:** In $O(m_H)$ time we can scan all edges and group them into color classes, compute the weight of each color class, and then find the up to three color classes of lowest weight. The same amount of time suffices for uncoloring all edges in those three color classes, thereby removing those colors from χ.

It remains to bound the time required for the **Repair** phase, denoted by $T(\mathit{Repair}\ H)$. We will prove that

$$\mathbb{E}[T(\mathit{Repair}\ H)] \;=\; O\left(\frac{W_H}{\Delta_H} \cdot \left(\frac{n}{\Delta_H} + \log n\right)\right), \tag{1}$$

and conclude that total expected time of the algorithm is

$$O\left(m_H + \frac{W_H}{\Delta_H} \cdot \left(\frac{n}{\Delta_H} + \log n\right)\right).$$

We shall bound the expected time for coloring the uncolored edges via Algorithm `Color-Edges`, where the uncolored edges are the ones that belong to the three color classes of minimum weight (We may assume w.l.o.g. that exactly three colors have been uncolored in H, out of a total of Δ_H+4 colors, by simply adding dummy color classes of weight 0). By an averaging argument, the total weight of the uncolored edges is bounded by

$$\frac{3}{\Delta_H+4}\cdot W_H = O\left(\frac{W_H}{\Delta_H}\right). \tag{2}$$

Let $PCL(H)$ be the set of all possible partial (Δ_H+1)-colorings of H, and for every coloring $\chi \in PCL(H)$ let $U(\chi)$ denote the number of edges of H that are uncolored by χ. In addition, let $PCL(H,l)$ be the set of all colorings $\chi \in PCL(H)$ with $U(\chi)=l$.

Note that the partial coloring obtained at the beginning of the **Repair** phase is not deterministic, and let χ_0 denote this random partial coloring. Thus, $U=U(\chi_0)$ is a random variable. Fix an arbitrary integer $l \geq 0$. Under the condition $U=l$, Algorithm `Color-Edges` consists of l iterations that color uncolored edges via Algorithm `Color-One-Edge`.

For every iteration $i=1,...,l$, let $\chi_i \in PCL(H,l+1-i)$ be the random partial coloring at the beginning of the iteration, let $e_i=(u_i,v_i)$ be the random uncolored edge chosen in line 1 of Algorithm `Color-One-Edge`, with $\mathrm{d}(u_i) \leq \mathrm{d}(v_i)$, let c_i be the random missing color of u_i chosen in line 4 of that algorithm, and let P_i be the maximal alternating path starting at u_i obtained in line 5 of that algorithm.

Each uncolored edge e_i is colored via a call to Algorithm `Color-One-Edge`, whose expected time is dominated (by Claim 16) by $O(\mathbb{E}[|P_i|]+\mathbb{E}[\mathrm{d}(u_i)])$. So the total expected time for coloring all the l uncolored edges under the condition $U=l$, namely $\mathbb{E}[T(\mathit{Repair}\ H) \mid U=l]$, satisfies

$$\mathbb{E}[T(\mathit{Repair}\ H) \mid U=l] \;=\; \sum_{i=1}^{l} O(\mathbb{E}[|P_i| \mid U=l]) + \sum_{i=1}^{l} O(\mathbb{E}[\mathrm{d}(u_i) \mid U=l]). \tag{3}$$

In App. C of the full version of our paper, we prove the following two claims, and use them to prove Eq. 1.

▷ Claim 22. For any integer $l \geq 0$, $\sum_{i=1}^{l} \mathbb{E}[\mathrm{d}(u_i) \mid U=l] = O\left(\frac{W_H}{\Delta_H}\right)$.

▷ Claim 23. For any integer $l \geq 0$, $\sum_{i=1}^{l} \mathbb{E}[|P_i| \mid U=l] = O\left(\frac{W_H}{\Delta_H}\cdot\left(\frac{n}{\Delta_H}+\log n\right)\right)$. ◀

In App. C of the full version of our paper, we prove the following lemma using Lemma 21.

▶ **Lemma 24.** *The expected runtime of Algorithm* `Recursive-Color-Edges` *on G is bounded by*

$$O\left(W\cdot\min\left\{\log n, \frac{\sqrt{n\log n}}{\Delta}\right\}+m\log n\right).$$

Remark. Claim 4 yields $W=O(m\alpha)$, hence the expected runtime of the algorithm is $O(\alpha m\cdot\min\{\log n, \frac{\sqrt{n\log n}}{\Delta}\}+m\log n)$, or equivalently, $O(\min\{m\Delta\cdot\log n, m\sqrt{n\log n}\}\cdot\frac{\alpha}{\Delta}+m\log n)$. It is straightforward to achieve the same bound on the running time (up to a logarithmic factor) with high probability. (As mentioned, we do not attempt to optimize polylogarithmic factors in this work.) Thus we derive the following corollary, which concludes the proof of Theorem 3.

▶ **Corollary 25.** *One can compute a* $(\Delta + 1)$*-coloring in any* n*-vertex* m*-edge graph of arboricity* α *and maximum degree* Δ *within a high probability runtime bound of*

$$O\left(\min\{m\Delta\log^2 n, m\sqrt{n}\log^{1.5} n\} \cdot \frac{\alpha}{\Delta} + m\log^2 n\right).$$

References

1 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.
2 Sepehr Assadi. Faster vizing and near-vizing edge coloring algorithms. *CoRR*, abs/2405.13371, 2024. `doi:10.48550/arXiv.2405.13371`.
3 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.
4 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Arboricity-dependent algorithms for edge coloring. *CoRR*, abs/2311.08367, 2023. To appear at SWAT'24. `doi:10.48550/arXiv.2311.08367`.
5 Béla Bollobás. *Graph theory*. Elsevier, 1982.
6 Yan Cao, Guantao Chen, Suyun Jiang, Huiqing Liu, and Fuliang Lu. Average degrees of edge-chromatic critical graphs. *Discrete Mathematics*, 342(6):1613–1623, 2019.
7 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. `doi:10.1137/0214017`.
8 Aleksander B. G. Christiansen, Eva Rotenberg, and Juliette Vlieghe. Sparsity-parameterised dynamic edge colouring. *CoRR*, abs/2311.10616, 2023. To appear at SWAT'24. `doi:10.48550/arXiv.2311.10616`.
9 Aleksander Bjørn Grodt Christiansen. The power of multi-step vizing chains. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1013–1026. ACM, 2023.
10 Marek Chrobak and Takao Nishizeki. Improved edge-coloring algorithms for planar graphs. *Journal of Algorithms*, 11(1):102–116, 1990.
11 Richard Cole and Łukasz Kowalik. New linear-time algorithms for edge-coloring planar graphs. *Algorithmica*, 50(3):351–368, 2008.
12 Stanley Fiorini. Some remarks on a paper by vizing on critical graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 77, pages 475–483. Cambridge University Press, 1975.
13 Harold N Gabow. Using euler partitions to edge color bipartite multigraphs. *International Journal of Computer & Information Sciences*, 5(4):345–355, 1976.
14 Harold N Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. Algorithms for edge-coloring graphs. *Technical Report*, 1985.
15 Dawit Haile. Bounds on the size of critical edge-chromatic graphs. *Ars Combinatoria*, 53:85–96, 1999.
16 Ian Holyer. The np-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.
17 Łukasz Kowalik. Edge-coloring sparse graphs with Δ colors in quasilinear time. *arXiv preprint arXiv:2401.13839*, 2024.
18 Jayadev Misra and David Gries. A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.
19 C St JA Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 1(1):12–12, 1964.
20 Daniel P Sanders and Yue Zhao. On the size of edge chromatic critical graphs. *Journal of Combinatorial Theory, Series B*, 86(2):408–412, 2002.

21 Corwin Sinnamon. A randomized algorithm for edge-colouring graphs in $o(m\sqrt{n})$ time. *CoRR*, abs/1907.03201, 2019. `arXiv:1907.03201`.

22 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30, 1964.

23 Douglas R Woodall. The average degree of an edge-chromatic critical graph ii. *Journal of Graph Theory*, 56(3):194–218, 2007.

24 Douglas R Woodall. The average degree of an edge-chromatic critical graph. *Discrete mathematics*, 308(5-6):803–819, 2008.

25 Xiao Zhou and Takao Nishizeki. Edge-coloring and f-coloring for various classes of graphs. In *Algorithms and Computation: 5th International Symposium, ISAAC'94 Beijing, PR China, August 25–27, 1994 Proceedings 5*, pages 199–207. Springer, 1994.