Arboricity-Dependent Algorithms for 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

The problem of edge coloring has been extensively studied over the years. Recently, this problem has received significant attention in the *dynamic setting*, where we are given a dynamic graph evolving via a sequence of edge insertions and deletions and our objective is to maintain an edge coloring of the graph.

Currently, it is not known whether it is possible to maintain a $(\Delta + O(\Delta^{1-\mu}))$-edge coloring in $\tilde{O}(1)$ update time, for any constant $\mu > 0$, where Δ is the maximum degree of the graph.[1] In this paper, we show how to efficiently maintain a $(\Delta + O(\alpha))$-edge coloring in $\tilde{O}(1)$ amortized update time, where α is the arboricty of the graph. Thus, we answer this question in the affirmative for graphs of sufficiently small arboricity.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic Algorithms, Graph Algorithms, Edge Coloring, Arboricity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.12

Related Version *Full Version*: `https://arxiv.org/abs/2311.08367`

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

Consider any graph $G = (V, E)$, with $n = |V|$ nodes and $m = |E|$ edges, and any integer $\lambda \geq 1$. A *(proper) λ-(edge) coloring* $\chi : E \to [\lambda]$ of G assigns a color $\chi(e) \in [\lambda]$ to each edge $e \in E$, in such a way that no two adjacent edges receive the same color. Our goal is to get a proper λ-coloring of G, for as small a value of λ as possible. It is easy to verify that any such coloring requires at least Δ colors, where Δ is the maximum degree of G. On the other hand, a textbook theorem by Vizing [13] guarantees the existence of a proper $(\Delta + 1)$-coloring in any input graph.

This work focuses on the edge coloring problem in the *dynamic setting*, where an extensive body of work has been devoted to this problem. Before describing our contributions, we first summarize the relevant state-of-the-art in the dynamic setting.

[1] We use $\tilde{O}(\cdot)$ to hide polylogarthmic factors.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 12; pp. 12:1–12:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Dynamic Edge Coloring. In the dynamic setting, the input graph G undergoes a sequence of updates (edge insertions/deletions), and throughout this sequence the concerned algorithm has to maintain a proper coloring of G. We wish to design a dynamic algorithm whose *update time* (time taken to process an update) is as small as possible. The edge coloring problem has received significant attention within the dynamic algorithms community in recent years. It is known how to maintain a $(2\Delta - 1)$-coloring in $O(\log \Delta)$ update time [2, 3], and Duan et al. [11] showed how to maintain a $(1 + \epsilon)\Delta$-coloring in $O(\log^8 n/\epsilon^4)$ update time when $\Delta = \Omega(\log^2 n/\epsilon^2)$. Subsequently, Christiansen [10] presented a dynamic algorithm for $(1+\epsilon)\Delta$-coloring with $O(\log^9 n \log^6 \Delta/\epsilon^6)$ update time, without any restriction on Δ. More recently, Bhattachrya et al. [5] showed how to maintain a $(1 + \epsilon)\Delta$-coloring in $O(\log^4(1/\epsilon)/\epsilon^9)$ update time when $\Delta \geq (\log n/\epsilon)^{\Theta((1/\epsilon)\log(1/\epsilon))}$. At present, no dynamic edge coloring algorithm is known with a sublinear in Δ *additive approximation* and with $\tilde{O}(1)$ update time. We summarize the following basic question that arises.

Is there a dynamic algorithm for maintaining a $(\Delta + O(\Delta^{1-\mu}))$-edge coloring with $\tilde{O}(1)$ update time, for any constant $\mu > 0$?

1.1 Our Contribution

We address the above question for the family of *bounded arboricity* graphs. Formally, a graph $G = (V, E)$ has *arboricity* (at most) α iff:

$$\left\lceil \frac{|E(G[S])|}{(|S| - 1)} \right\rceil \leq \alpha \text{ for every subset } S \subseteq V \text{ of size } |S| \geq 2,$$

where $G[S]$ denotes the subgraph of G induced by S and $E(G[S])$ denotes the edge-set of $G[S]$. It is easily verified that the arboricity of any graph is upper bounded by its maximum degree. There are many instances of graphs, however, with very high maximum degree but low arboricity.[2] Intuitively, a graph with low arboricity is *sparse everywhere.* Every graph excluding a fixed minor has $O(1)$ arboricity, thus the family of constant arboricity graphs contains bounded treewidth and bounded genus graphs, and specifically, planar graphs. More generally, graphs of bounded (not necessarily constant) arboricity are of importance, as they arise in real-world networks and models, such as the world wide web graph, social networks and various random distribution models.

We now summarize our main result.

▶ **Theorem 1.** *There is a deterministic dynamic algorithm for maintaining a* $(\Delta + (4 + \epsilon)\alpha)$*-edge coloring of an input dynamic graph with maximum degree* Δ *and arboricity* α*, with* $O(\log^6 n/\epsilon^6)$ *amortized update time and* $O(\log^4 n/\epsilon^5)$ *amortized recourse.*[3]

Thus, Theorem 1 addresses the above question in the affirmative, for all dynamic graphs with arboricity at most $O(\Delta^{1-\mu})$, for any constant $\mu > 0$.

An important feature of our dynamic algorithm is that it is *adaptive* to changes in the values of Δ and α over time: At each time-step t, we (explicitly) maintain a proper edge coloring of the input graph G using the colors $\{1, \ldots, \Delta_t + (4 + \epsilon)\alpha_t\}$, where Δ_t and α_t are respectively the maximum degree and arboricity of G at time t.

[2] Think of a star graph on n nodes. It has $\Delta = n - 1$ but $\alpha = 1$.

[3] A dynamic algorithm has an *amortized update time* (respectively, *amortized recourse*) of $O(\lambda)$, if, starting with an empty graph, the total runtime (resp., number of output changes) to handle any sequence of T updates is $O(T \cdot \lambda)$.

Before giving our full dynamic algorithm, we give a simpler "warmup" dynamic algorithm, where we assume access to values α and Δ such that $\alpha_t \leq \alpha$ and $\Delta_t \leq \Delta$ at each time-step t. In this setting, we can maintain a $(\Delta + (4 + \epsilon)\alpha)$-edge coloring with $O(\log^2 n \log \Delta/\epsilon^2)$ amortized update time and $O(\log n/\epsilon)$ worst-case recourse. As an immediate corollary of our "warmup" dynamic algorithm, we also get the following structural result, which should be contrasted with the lower bound of [7] for extending partial colorings, which shows that there exist n-node graphs of maximum degree Δ and $(\Delta + c)$-edge colorings on those graphs (for any $c \in [1, \Delta/3]$), such that extending these colorings to color some uncolored edge requires changing the colors of $\Omega(\Delta \log(cn/\Delta)/c)$ many edges.

▶ **Corollary 2.** *Let $G = (V, E)$ be a graph with maximum degree Δ and arboricity α, and let χ be a $(\Delta + (2 + \epsilon)\alpha)$-edge coloring of G. Then, given any uncolored edge $e \in E$, we can extend the coloring χ so that e is now colored by only changing the colors of $O(\log n/\epsilon)$ many edges.*

Independent Work. In independent and concurrent work, Christiansen, Rotenberg and Vlieghe also obtain a deterministic dynamic algorithm that maintains a $(\Delta + O(\alpha))$-edge coloring in $\tilde{O}(1)$ amortized update time [9].

1.2 Our Techniques

At a high level, our algorithm can be interpreted as a dynamization of a simple static algorithm that computes a $(\Delta+O(\alpha))$-edge coloring of a graph G, which can be implemented to run in near-linear time in the static sequential model of computation.[4] This algorithm is similar to the classic greedy algorithm for $(2\Delta - 1)$-edge coloring, which simply scans through all edges of the graph in an arbitrary order and, while scanning any edge e, assigns e an arbitrary color in $[2\Delta - 1]$ that has not been already assigned to one of its adjacent edges. Since e has at most $2\Delta - 2$ adjacent edges, such a color must always exist. This static algorithm does something quite similar – the difference is that it computes a "good" ordering of the edges in G instead of using an arbitrary ordering, which allows it to use fewer colors. More specifically, it repeatedly identifies a vertex of minimum degree in G, colors an edge incident on in, and removes that edge from the graph. For the sake of completeness, we include this algorithm and its analysis in Appendix A of the full version of our paper. We remark that a variant of this algorithm appears in [1], which considers the distributed model of computation.

To highlight the main conceptual insight underlying our approach, we describe the simpler case where Δ and α are *fixed* values (known to the algorithm in advance) that respectively give upper bounds on the maximum degree and arboricity of the input graph at all times. We sketch below how to maintain a $(\Delta + O(\alpha))$-coloring in $\tilde{O}(1)$ update time in this setting. Note that this directly implies a near-linear time static algorithm for $(\Delta + O(\alpha))$-coloring.[5] We later outline (Section 1.2.1) how we extend our dynamic algorithm to handle the scenario where Δ and α change over time.

Our starting point is a well-known "peeling process", which leads to a standard decomposition of an input graph $G = (V, E)$ with arboricity at most α [8]. The key observation is that *any* induced subgraph of G has average degree at most 2α.[6] Fix any constant $\gamma > 1$.

[4] Recently, [4] and [12] considered edge coloring on low arboricity graphs in the static setting, but for the problems of $\Delta + 1$ and Δ coloring respectively.

[5] Indeed, we can compute Δ and a good approximation of α in linear time, and then simply insert the edges in the input graph into the dynamic algorithm one after another.

[6] Indeed, for any subset $S \subseteq V$, the average degree of $G[S]$ is given by: $2 \cdot |E(G[S])|/|S| \leq 2\alpha$.

This motivates the following procedure, which runs for $L = \Theta_\gamma(\log n)$ rounds.

Initially, during round 1, we set $Z_1 := V$. Subsequently, during each round $i \in \{2, \ldots, L\}$, we find the set of nodes $S \subseteq Z_{i-1}$ that have degree $> 2\gamma\alpha$ in $G[Z_{i-1}]$, and set $Z_i := S$.

Consider any given round $i \in [L]$ during the above procedure. Since the subgraph $G[Z_{i-1}]$ has average degree at most 2α, it follows that at most a $1/\gamma$ fraction of the nodes in $G[Z_{i-1}]$ have degree more than $2\gamma\alpha$. In other words, we get $|Z_{i+1}| \leq |Z_i|/\gamma$, and hence after L iterations we would have $Z_L = \varnothing$. Bhattacharya et al. [6] showed how to maintain this decomposition dynamically with $\tilde{O}(1)$ amortized update time, provided that $\gamma > 2$.

Now, our dynamic $(\Delta + O(\alpha))$-coloring algorithm works as follows. Suppose that we are currently maintaining a valid coloring, along with the above decomposition. Upon receiving an update (edge insertion/deletion), we first run the dynamic algorithm of [6], which adjusts the decomposition $Z_1 \supseteq \cdots \supseteq Z_L$, in amortized $\tilde{O}(1)$ time. If the update consisted of an edge deletion, then we do not need to do anything else beyond this point, since the existing coloring continues to remain valid. We next consider the more interesting case, where the update consisted of the insertion of an edge (say) (u, v).

Let $i \in [L]$ be the largest index such that $(u, v) \in E(G[Z_i])$. Then there must exist some endpoint $x \in \{u, v\}$ that belongs to $Z_i \setminus Z_{i+1}$. W.l.o.g., let u be that endpoint. Since $u \in Z_i \setminus Z_{i+1}$, it follows that the node u has degree at most $2\gamma\alpha$ in $G[Z_i]$. Also, the node v trivially has degree at most Δ in G. Let $E_{(u,v)} \subseteq E$ denote the set of edges $e' \in E$ that belong to one of the following two categories: (I) e' is incident on u and lies in $G[Z_i]$, (II) e' is incident on v. We conclude that $|E_{(u,v)}| \leq \Delta + 2\gamma\alpha$. Thus, if we have a palette of at least $\Delta + 2\gamma\alpha + 1 = \Delta + \Theta(\alpha)$ colors, then there must exist a free color in that palette which is not assigned to any edge in $E_{(u,v)}$. Let c be that free color. Using standard binary search data structures, such a color c can be identified in $\tilde{O}(1)$ time [3]. We assign the color c to the edge (u, v). This can potentially create a conflict with some other adjacent edge $e'' \in E$ (which might already have been assigned the color c).

However, it is easy to see that such an edge e'' must be incident on u, i.e., $e'' = (u, y)$ for some $y \in V$, and there must exist some index $i_y < i$ such that $y \in Z_{i_y} \setminus Z_{i_y+1}$. We then uncolor the edge e'', set $i \leftarrow i_y$, and recolor e'' recursively using the same procedure described above. Since after each recursive call, the value of the index i decreases by at least one, this can go on at most L times. This leads to an overall update time of $L \cdot \tilde{O}(1) = \tilde{O}(1)$. See Section 3 for details.

1.2.1 Handling the scenario where Δ and α change over time

We now outline how we deal with changing values of Δ and α. Let α_t and Δ_t respectively denote the arboricity and maximum degree of the input graph G at the current time-step t. We need to overcome two technical challenges.

(i) The “warmup” algorithm described above works correctly only if it uses a parameter $\alpha \simeq \alpha_t$ to construct the decomposition of G. Informally, if α is too small w.r.t. α_t, then the number of iterations L required to construct the decomposition will become huge (possibly infinite, if we aim at achieving $Z_L = \varnothing$), and this in turn would blow up the update time of the algorithm. In contrast, if α is too large compared to α_t, then the algorithm would be using too many colors in its palette.

(ii) After the deletion of an edge e, the arboricity α and the maximum degree Δ of G might decrease. If either parameter drops by a significant amount (across some batch of updates), then we might have to recolor a significant number of edges to ensure that we are still only using $\Delta + O(\alpha)$ many colors, potentially leading to a prohibitively large update time.

To deal with challenge (i), we generalize the notion of graph decomposition to that of a *decomposition system*. At a high level, a decomposition system is just a collection of graph decompositions, where the relevant parameter across the decompositions is discretized into powers of $(1 + \epsilon)$. This ensures that no matter what the value of α is at the present moment, there is always some decomposition in our system that we can use to extend the coloring. Finally, to deal with challenge (ii), we ensure that the color of each edge satisfies certain *local constraints*, similar to the constraints used to give efficient dynamic algorithms in [3, 10]. After the deletion of an edge, we can just uncolor the edges that violate those local constraints, and then recolor them using the decomposition system. However, since the constraints on an edge e depend not just on the degrees of its endpoints but also on the decomposition system, we have to take extra care to ensure that these decompositions don't change too much between updates. See Section 4 for details.

1.3 Roadmap

The rest of the paper is organized as follows. Section 2 introduces the relevant preliminary concepts and notations. This is followed by Section 3, which contains our warmup dynamic algorithm for fixed α. In Section 4, we present our dynamic algorithm in its full generality. Appendix B in the full version of our paper gives the full details of the relevant data structures used by our algorithms.

2 Preliminaries

In this section, we define the notations used throughout our paper and describe the notion of *graph decompositions*, which are at the core of our algorithms. We then provide a simple extension of these graph decompositions, which we use as a central component in our final dynamic algorithm.

2.1 The Dynamic Setting

In the dynamic setting, we have a graph $G = (V, E)$ that undergoes updates via a sequence of intermixed edge insertions and deletions. Our task is to design an algorithm to explicitly maintain an edge coloring χ of G as the graph is updated. We assume that the graph G is initially empty, i.e. that the graph G is initialized with $E = \varnothing$. The *update time* of such an algorithm is the time it takes to handle an update, and its *recourse* is the number of edges that change colors while handling an update. More precisely, we say that an algorithm has a worst-case update time of λ if it takes at most λ time to handle an update, and an *amortized* update time of λ if it takes at most $T \cdot \lambda$ time to handle any arbitrary sequence of T updates (starting from the empty graph). Similarly, we say that an algorithm has a worst-case recourse of λ if it changes the colors of at most λ edges while handling an update, and an *amortized* recourse of λ if it changes the colors of at most $T \cdot \lambda$ edges while handling any arbitrary sequence of T updates (starting from the empty graph).

2.2 Notation

Let $G = (V, E)$ be an undirected, unweighted n-node graph. Given an edge set $S \subseteq E$, we denote by $G[S]$ the graph (V, S), and given a node set $A \subseteq V$, we denote by $G[A]$ the subgraph induced by A, namely $(A, \{(u, v) \in E \mid u, v \in A\})$. Given a node $u \in V$ and a subgraph H of G, we denote by $N_H(u)$ the set of edges in H that are incident on u, and by $\deg_H(u)$ the degree of u in H. For an edge (u, v), we define $N_H(u, v)$ to be $N_H(u) \cup N_H(v)$. When we are considering the entire graph G, we will often omit the subscripts in $N_G(\cdot)$ and $\deg_G(\cdot)$ and just write $N(\cdot)$ and $\deg(\cdot)$.

2.3 Graph Decompositions

A central ingredient in our dynamic algorithm is the notion of (β, d, L)-*decomposition*, defined by Bhattacharya et al. [6].

▶ **Definition 3.** *Given a graph $G = (V, E)$, $\beta \geq 1$, $d \geq 0$, and a positive integer L, a (β, d, L)-decomposition of G is a sequence $(Z_1, \dots, Z_L)$ of node sets, such that $Z_L \subseteq \cdots \subseteq Z_1 = V$ and*

$$Z_{i+1} \supseteq \{u \in Z_i \mid \deg_{G[Z_i]}(u) > \beta d\} \text{ and } Z_{i+1} \cap \{u \in Z_i \mid \deg_{G[Z_i]}(u) < d\} = \varnothing$$

hold for all $i \in [L-1]$.

Given a (β, d, L)-decomposition $(Z_1, \dots, Z_L)$ of $G = (V, E)$, we abbreviate $G[Z_i]$ as G_i for all i, and for all $u \in V$, we abbreviate $\deg_{G_i}(u)$ as $\deg_i(u)$ and $N_{G_i}(u)$ as $N_i(u)$. We define $V_i := Z_i \setminus Z_{i+1}$ for all $i \in [L-1]$, and $V_L := Z_L$. We say that V_i is the i^{th} *level* of the decomposition, and define the *level* $\ell(u)$ of any node $u \in V_i$ as $\ell(u) := i$. We define $\deg^+(u) := \deg_{\ell(u)}(u)$ and $N^+(u) := N_{\ell(u)}(u)$ for $u \in V$. Given an edge $e = (u, v)$, we define the level $\ell(e)$ of e as $\ell(e) := \min\{\ell(u), \ell(v)\}$. Note also that for all $u \in V \setminus V_L$, $\deg^+(u) \leq \beta d$. However, given some $u \in V_L$, $\deg^+(u)$ may be much larger than βd, which motivates the following useful fact concerning such decompositions.

▶ **Lemma 4** ([6]). *Let $G = (V, E)$ be an arbitrary graph with arboricity α, let β, ϵ, d be any parameters such that $\beta \geq 1$, $0 < \epsilon < 1$, $d \geq 2(1+\epsilon)\alpha$, and let $L = 2 + \lceil \log_{(1+\epsilon)} n \rceil$. Then for any (β, d, L)-decomposition $(Z_1, ..., Z_L)$ of G, it holds that $Z_L = \varnothing$.*

Proof. Let $(Z_1, ..., Z_L)$ be a (β, d, L)-decomposition of G satisfying the conditions of the lemma. Let i be an arbitrary index in $[L-1]$. Since the arboricity of G_i is at most α, the average degree in G_i is at most 2α. On the other hand, by definition, the degree of any node in Z_{i+1} in the graph G_i is at least $d \geq 2(1+\epsilon)\alpha$. It follows that

$$2(1+\epsilon)\alpha|Z_{i+1}| \leq \sum_{u \in Z_{i+1}} \deg_i(u) \leq \sum_{u \in Z_i} \deg_i(u) \leq 2\alpha|Z_i|,$$

and hence $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$. Inductively, we obtain $|Z_L| \leq (1+\epsilon)^{1-L}|Z_1| \leq 1/(1+\epsilon) < 1$, yielding $Z_L = \varnothing$. ◀

Orienting the Edges. For our purposes, it will be useful to think of a decomposition of G as inducing an *orientation* of the edges. In particular, given an edge $e = (u, v)$, *we orient the edge from the endpoint of lower level towards the endpoint of higher level.* If the two endpoints have the same level, we orient the edge arbitrarily. We write $u \prec v$ to denote that the edge e is oriented from u to v. Note that $\deg^+(u)$ is an upper bound on the out-degree of u with respect to this orientation of the edges.

Dynamic Decompositions. Bhattacharya et al. give a deterministic fully dynamic data structure that can be used to explicitly maintain a (β, d, L)-decomposition of a graph $G = (V, E)$ under edge updates with small amortized update time. This algorithm also has small amortized recourse, where the recourse of an update is defined as the number of edges that change level following the update. The following theorem, from Section 4.1 of [6], will be used as a black box in our dynamic algorithm.

▶ **Proposition 5** ([6]). *For any constant $\beta \geq 2 + 3\epsilon$, there is a deterministic fully-dynamic algorithm that maintains a (β, d, L)-decomposition of a graph $G = (V, E)$ with amortized update time and amortized recourse both bounded by $O(L/\epsilon)$.*

It is straightforward to modify this dynamic algorithm to explicitly maintain the orientation of the edges that we described above without changing its asymptotic behavior. Furthermore, we can assume that the orientation of an edge changes *only* when it changes level.

2.4 Graph Decomposition Systems

In order for our dynamic algorithm to be able to deal with dynamically changing arboricity α, we will need to give a slight generalization of Definition 3, which we refer to as a *decomposition system*. Intuitively, this will enable us to maintain multiple decompositions, one for each "guess" of the arboricity, allowing us to use whichever decomposition is most appropriate to modify the edge coloring while handling an update.

▶ **Definition 6.** *Given a graph $G = (V, E)$, $\beta \geq 1$, a sequence $(d_j)_{j\in[K]}$ such that $d_j \geq 0$, and a positive integer L, a $(\beta, (d_j)_{j\in[K]}, L)$-decomposition system of G is a sequence $(Z_{i,j})_{i\in[L],j\in[k]}$ of node sets, where for each $j \in [K]$, $(Z_{i,j})_{i\in[L]}$ is a (β, d_j, L)-decomposition of G.*

Given a $(\beta, (d_j)_{j\in[K]}, L)$-decomposition system of $G = (V, E)$, we denote the graph $G[Z_{i,j}]$ by $G_{i,j}$, $\deg_{G_{i,j}}(u)$ by $\deg_{i,j}(u)$, and $N_{G_{i,j}}(u)$ by $N_{i,j}(u)$ for $u \in V$. We say that $(Z_{i,j})_i$ is the j^{th} *layer* of the decomposition system. We denote by $\ell_j(u)$ the level of node u in the decomposition $(Z_{i,j})_i$ and define $\deg^+_j(u) := \deg_{\ell_j(u),j}(u)$ and $N^+_j(u) := N_{\ell_j(u),j}(u)$ for $u \in V$.

Given a node u, we define the *layer* of u as $\mathcal{L}(u) = \min\{j \in [K] \mid \ell_j(u) < L\}$. Given an edge $e = (u, v)$, we define the layer of e as $\mathcal{L}(e) = \min\{\mathcal{L}(u), \mathcal{L}(v)\}$. We denote the orientation of the edges induced by the decomposition $(Z_{i,j})_i$ by $\prec_j$.

We can use the data structure from Proposition 5 to dynamically maintain a decomposition system, giving us the following proposition. In this context, we define the recourse of an update to be the number of edges that change levels in *some* layer.

▶ **Proposition 7.** *For any constant $\beta \geq 2 + 3\epsilon$, there is a deterministic fully dynamic algorithm that maintains a $(\beta, (d_j)_{j\in[K]}, L)$-decomposition system of a graph $G = (V, E)$ with amortized update time and amortized recourse $O(KL/\epsilon)$.*

As before, we assume that the orientation of an edge e with respect to $\prec_j$ changes only when $\ell_j(e)$ changes.

3 A Warmup Dynamic Algorithm (for Fixed α)

We now turn our attention towards designing an algorithm that can *dynamically* maintain a $(\Delta + O(\alpha))$-edge coloring of the graph G as it changes over time. A starting point for creating such an algorithm is the static algorithm that we outline in Section 1.2. Unfortunately, the highly sequential nature of this algorithm makes it very challenging to dynamize directly, as it is not clear how to efficiently maintain the output in the dynamic setting. In order to overcome this obstacle, we use the notion of graph decompositions (see Section 2.3). Informally, these graph decompositions can be interpreted as an "approximate" version of the sequence in which the static algorithm colors the edges in the graph – where instead of peeling off a node with smallest degree one at a time, we peel off large batches of nodes with sufficiently small degrees simultaneously. This leads to a "more robust" structure that can be maintained dynamically in an efficient manner.

Let $G = (V, E)$ be a dynamic graph that undergoes updates via edge insertions and deletions. In this section, we work in a simpler setting where we assume that we are given an α and are guaranteed that the maximum arboricity of the graph G remains at most α throughout the entire sequence of updates. We then give a deterministic fully dynamic algorithm that maintains a $(\Delta + O(\alpha))$-edge coloring of G, where Δ is an upper bound on the maximum degree of G at any point throughout the entire sequence of updates.[7] Without dealing with implementation details, we show that it achieves $\tilde{O}(1)$ worst-case recourse per update. In Section 4, we extend our result to the setting where Δ and α are not bounded and show how to maintain a $(\Delta + O(\alpha))$-edge coloring of G where α and Δ are the *current* arboricity and maximum degree of G respectively and change over time.

3.1 Algorithm Description

For the rest of this section, fix some constants ϵ, β, and L such that: $0 < \epsilon < 1$, $\beta = 2 + 3\epsilon$, $L = 2 + \lceil \log_{1+\epsilon} n \rceil$. At a high level, our algorithm works by dynamically maintaining a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_{i=1}^{L}$ of the graph G by using Proposition 5. During an update, our algorithm first updates the decomposition $(Z_i)_i$, and then uses this decomposition to find a path of length at most L such that, by only changing the colors assigned to the edges in this path, it can update the coloring to be valid for the updated graph. Since $L = \tilde{O}(1)$, this immediately implies the worst-case recourse bound. Algorithm 1 gives the procedure that we call to initialize our data structure, creating a decomposition of the empty graph, and Algorithms 2 and 3 give the procedures called when handling insertions and deletions respectively.

Algorithm 1 $\textsc{Initialize}(G, \alpha)$.

Input: An empty graph $G = (V, \varnothing)$ and a parameter α

Create a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_{i \in [L]}$ of G

Algorithm 2 $\textsc{Insert}(e)$.

Input: An edge e to be inserted into G

Insert the edge e into G

$\chi(e) \leftarrow \perp$

Update the $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_i$ of G

$\textsc{ExtendColoring}(e, (Z_i)_i)$

[7] Note that the algorithm needs prior knowledge of α, but not Δ.

Algorithm 3 DELETE(e).

Input: An edge e to be deleted from G

Delete the edge e from G

$\chi(e) \leftarrow \perp$

Update the $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_i$ of G

Algorithm 4 EXTENDCOLORING($e, (Z_i)_i$).

Input: An uncolored edge e and a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_i$ of G

$S \leftarrow \{e\}$

while $S \neq \varnothing$ **do**

  Let $f = (u, v)$ be any edge in S where $u \prec v$

  $C_u^+ \leftarrow \chi(N^+(u))$

  $C_v \leftarrow \chi(N(v))$

  Set c to any element in $[|C_u^+| + |C_v| + 1] \setminus (C_u^+ \cup C_v)$

  if $c \in \chi(N(u))$ **then**

    Let f' be the edge in $N(u)$ with $\chi(f') = c$

    $\chi(f') \leftarrow \perp$ and $S \leftarrow S \cup \{f'\}$

  $\chi(f) \leftarrow c$ and $S \leftarrow S \setminus \{f\}$

The following theorem, which we prove next, summarizes the behavior of our warmup dynamic algorithm.

▶ **Theorem 8.** *The warmup dynamic algorithm is deterministic and, given a sequence of updates for a dynamic graph G and a value α such that the arboricity of G never exceeds α, maintains a $(\Delta + (4+\epsilon)\alpha)$-edge coloring, where Δ is the maximum degree of G throughout the entire sequence of updates. The algorithm has $O(\log n/\epsilon)$ worst-case recourse per update and $O(\log^2 n \log \Delta/\epsilon^2)$ amortized update time.*

3.2 Analysis of the Warmup Algorithm

We now show that the warmup algorithm maintains a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring and has a worst-case recourse of at most $L = O(\log n/\epsilon)$ per update.[8]

▶ **Lemma 9.** *Let $G = (V, E)$ be a graph with maximum degree at most Δ and arboricity at most α. Let e be an edge in G, $(Z_i)_i$ a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition of G and χ a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of $G - e$. Then running* EXTENDCOLORING$(e, (Z_i)_i)$*:*

1. *changes the colors of at most L edges in G, and*
2. *turns χ into a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of G.*

Proof. We first prove (1). Let e_i denote the edge that is uncolored at the start of the i^{th} iteration of the while loop as we run the procedure. Let $\ell(e_i)$ denote the minimum of the level of both of its endpoints. Clearly $\ell(e_i) \leq L$ since this is the highest level and $\ell(e_i) \geq 1$ for all i since this is the lowest level. Suppose the while loop iterates at least i times for some integer $i \geq 2$. Let $e_{i-1} = (u, v)$ where $u \prec v$, and hence $\ell(u) \leq \ell(v)$ (see Section 2.3). Since $e_i \in N(u)$ during iteration $i - 1$ but $\chi(e_i) \notin \chi(N^+(u))$, we have that $e_i \notin N^+(u)$, and hence the endpoint of e_i that is not u appears in a level strictly below the level of u, so

[8] Note that $2\beta(1+\epsilon)\alpha = (4 + O(\epsilon))\alpha$.

$\ell(e_i) < \ell(e_{i-1})$. It follows that $1 \leq \ell(e_i) \leq L + 1 - i$, so the while loop iterates at most L times. For (2), note that if we let $e_i = (u, v)$ where $u \prec v$, then $|C_u^+| = \deg^+(u) - 1$ and $|C_v| = \deg(v) - 1$, so

$$|C_u^+| + |C_v| + 1 \leq \deg^+(u) + \deg(v) - 1 \leq \Delta + 2\beta(1 + \epsilon)\alpha,$$

and so the procedure never assigns any e_i a color larger than $\Delta + 2\beta(1 + \epsilon)\alpha$. Since we know from (1) that the procedure terminates after at most L iterations, after which every edge in the graph is colored, and χ was a $(\Delta + 2\beta(1 + \epsilon)\alpha)$-edge coloring of the graph $G - e_1$ at the start of the procedure, it follows by induction that after the procedure terminates χ assigns each edge in G a color from $[\Delta + 2\beta(1 + \epsilon)\alpha]$. Furthermore, our algorithm can only terminate if this assignment forms a valid edge coloring. Hence, χ is a $(\Delta + 2\beta(1 + \epsilon)\alpha)$-edge coloring of G. ◀

▶ **Lemma 10.** *The warmup algorithm maintains a $(\Delta + 2\beta(1 + \epsilon)\alpha)$-edge coloring of the graph.*

Proof. We prove this by induction. Since G is initially empty, the empty map is trivially a coloring of G. Let $\lambda = \Delta + 2\beta(1 + \epsilon)\alpha$. Suppose χ is a λ-edge coloring of G after the i^{th} update. If the $i + 1^{th}$ update is a deletion, χ is still a λ-edge coloring of the updated graph and we are done. If the $i + 1^{th}$ update is an insertion, then we run Algorithm 4 in order to update χ. By part (2) of Lemma 9, it follows that χ is a λ-edge coloring of the updated graph once the procedure terminates. ◀

▶ **Lemma 11.** *The warmup algorithm changes the colors of at most L edges while handling an update.*

Proof. While handling the deletion of an edge e, our algorithm uncolors the edge e and does not change the color of any other edge. While handling the insertion of an edge e, our algorithm only changes the colors of edges while handling the call to $\textsc{ExtendColoring}(e, (Z_i)_i)$. By part (1) of Lemma 9, this changes the colors of at most L edges. ◀

In the full version of our paper, we prove the following lemma.

▶ **Lemma 12.** *The warmup algorithm has an amortized update time of $O(\log^2 n \log \Delta/\epsilon^2)$.*

We also note that Corollary 2 follows immediately from Lemma 9. In particular, if we set $\beta = 1$, by Lemma 4, the proof Lemma 9 still holds. Hence, we can use $\textsc{ExtendColoring}$ along with *any* $(1, 2(1+\epsilon)\alpha, L)$-decomposition of G in order to extend any $(\Delta+2(1+\epsilon)\alpha)$-edge coloring χ with an uncolored edge e so that the edge e is now colored by only changing the colors of $O(\log n/\epsilon)$ many edges.

4 The Dynamic Algorithm

We now describe our full dynamic algorithm and show that it maintains a $(\Delta + O(\alpha))$-edge coloring of the graph. We then use Proposition 7 to show that we can get $\tilde{O}(1)$ amortized recourse. In Appendix B of the full version of our paper, we describe the relevant data structures and use them to implement our algorithm to get $\tilde{O}(1)$ amortized update time.

4.1 Algorithm Description

In order to describe our algorithm, we fix some constant ϵ such that $0 < \epsilon < 1$ and set $\beta = 2 + 3\epsilon$, $L = 2 + \lceil \log_{1+\epsilon} n \rceil$. Let $\tilde{\alpha}_j := (1 + \epsilon)^{j-1}$ and note that, for any n-node graph G with arboricity α, $\tilde{\alpha}_1 = 1 \leq \alpha \leq n < \tilde{\alpha}_L$.

Informal Description. Our algorithm works by maintaining the invariant that each edge $e = (u, v)$ receives a color in the set $[\deg(v) + O(\tilde{\alpha}_{\mathcal{L}(e)})]$, where $u \prec_{\mathcal{L}(e)} v$. Since $\deg(v) \leq \Delta$ and $\tilde{\alpha}_{\mathcal{L}(e)} = O(\alpha)$ (see Lemma 15), it follows that the algorithm uses at most $\Delta + O(\alpha)$ many colors. When an edge is inserted or deleted, this may cause some $\tilde{O}(1)$ many edges to violate the invariant. We begin by first identifying all such edges and uncoloring them. We then update the decomposition system maintained by our algorithm, which may again cause some $\tilde{O}(1)$ many edges (on average) to violate the invariant. We again identify and uncolor all such edges. We now want to color each of the uncolored edges, while ensuring that we satisfy this invariant at all times. We do this by using the decomposition system maintained by our algorithm: we take an uncolored edge $f = (u, v)$ such that $u \prec_{\mathcal{L}(f)} v$ and assign it a color c that is *not* assigned to any of the edges in $N^{+}_{\mathcal{L}(f)}(u)$ or $N(v)$. If there is an edge f' adjacent to f that is also colored with c, we uncolor this edge. We repeat this process iteratively until all edges are colored. We can show that: (1) there are at most $\deg(v) + O(\tilde{\alpha}_{\mathcal{L}(f)})$ many edges in $N^{+}_{\mathcal{L}(f)}(u) \cup N(v)$, and hence we can find such a c in the palette $[\deg(v) + O(\tilde{\alpha}_{\mathcal{L}(f)})]$, and (2) if there is such an edge f' adjacent to f that is also colored with c, then either $\ell_{\mathcal{L}(f')}(f') < \ell_{\mathcal{L}(f)}(f)$ or $\mathcal{L}(f') < \mathcal{L}(f)$, allowing us to carry out a potential function argument that shows that the process terminates with all edges colored after $\tilde{O}(1)$ iterations on average, giving us an amortized recourse bound.

Formal Description. The following pseudo-code gives a precise formulation of our algorithm.

Algorithm 5 INITIALIZE(G).

Input: An empty graph $G = (V, \varnothing)$

Create a $(\beta, (2(1+\epsilon)\tilde{\alpha}_j)_{j\in[L]}, L)$-decomposition system $(Z_{i,j})_{i,j\in[L]}$ of G

Algorithm 6 INSERT(e).

Input: An edge e to be inserted into G

Insert the edge e into G

$S \leftarrow$ UPDATEDECOMPOSITIONS(e)

$\chi(f) \leftarrow \perp$ for all $f \in S$

EXTENDCOLORING(S)

Algorithm 7 DELETE(e).

Input: An edge e to be deleted from G

Delete the edge e from G

$S \leftarrow \varnothing$

for $v \in e$ **do**

  $S \leftarrow S \cup \{f = (u, v) \in N(v) \mid u \prec_{\mathcal{L}(f)} v$ and $\chi(f) > \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}\}$

$S \leftarrow S \cup$ UPDATEDECOMPOSITIONS(e)

$\chi(f) \leftarrow \perp$ for all $f \in S$

EXTENDCOLORING(S)

Algorithm 8 UPDATEDECOMPOSITIONS(e).

Input: The edge e that has been inserted/deleted from G

Update the decomposition system $(Z_{i,j})_{i,j}$

Let $S' \subseteq E$ be the set of all edges whose level changes in some layer

return S'

Algorithm 9 $\textsc{ExtendColoring}(S)$.

Input: A set S of uncolored edges
while $S \neq \varnothing$ **do**
  Let $f = (u, v)$ be any edge in S where $u \prec_{\mathcal{L}(f)} v$
  $C_u^+ \leftarrow \chi(N^+_{\mathcal{L}(f)}(u))$
  $C_v \leftarrow \chi(N(v))$
  Let c be any element in $[|C_u^+| + |C_v| + 1] \setminus (C_u^+ \cup C_v)$
  if $c \in \chi(N(u))$ **then**
    Let f' be the edge in $N(u)$ with $\chi(f') = c$
    $\chi(f') \leftarrow \perp$ and $S \leftarrow S \cup \{f'\}$
  $\chi(f) \leftarrow c$ and $S \leftarrow S \setminus \{f\}$

The following theorem, which we prove next, summarizes the behavior of our full dynamic algorithm.

▶ **Theorem 13.** *The dynamic algorithm is deterministic and, given a sequence of updates for a dynamic graph G, maintains a $(\Delta + (4+\epsilon)\alpha)$-edge coloring, where Δ and α are the dynamically changing maximum degree and arboricity of G, respectively. The algorithm has $O(\log^4 n/\epsilon^5)$ amortized recourse per update and $O(\log^5 n \log \Delta/\epsilon^6)$ amortized update time.*[9]

We split the proof of Theorem 13 into two parts. In Section 4.2, we show that our dynamic algorithm maintains a $(\Delta + 2\beta(1+\epsilon)^2\alpha)$-edge coloring and has an amortized recourse of $O(\log^4 n/\epsilon^5)$.[10] In Appendix B of the full version of our paper, we describe the data structures used by our algorithm, before showing how to use them in order to get $O(\log^5 n \log \Delta/\epsilon^6)$ amortized update time.

4.2 Analysis of the Dynamic Algorithm

For the rest of Section 4.2, fix a dynamic graph $G = (V, E)$, and a $(\beta, (2(1+\epsilon)\tilde{\alpha}_j)_j, L)$-decomposition system $\mathcal{Z} = (Z_{i,j})_{i,j}$ of G. Recall that ϵ is a fixed constant with $0 < \epsilon < 1$, and that $\beta = 2 + 3\epsilon$, $L = 2 + \lceil \log_{1+\epsilon} n \rceil$.

We begin with the following simple observations.

▶ **Lemma 14.** *For all nodes $u \in V$, we have that $\mathcal{L}(u) \le j^\star$, where $j^\star \in [L]$ is the unique value such that $\alpha \le \tilde{\alpha}_{j^\star} < (1+\epsilon)\alpha$.*

Proof. By Lemma 4, we know that $Z_{L,j^\star} = \varnothing$. Hence, $\mathcal{L}(u) \le j^\star$ for every node $u \in V$. ◀

▶ **Corollary 15.** *For all edges $e \in E$, we have that $\tilde{\alpha}_{\mathcal{L}(e)} < (1+\epsilon)\alpha$.*

We now define the notation of a *good* edge coloring. In such an edge coloring, the colors satisfy certain locality constraints, which makes it easier to maintain dynamically.

▶ **Definition 16.** *Given an edge coloring χ of the graph G, we say that χ is a* good *edge coloring of G with respect to the decomposition system $\mathcal{Z}$ if and only if for every edge $e = (u, v) \in E$ such that $\chi(e) \neq \perp$ and $u \prec_{\mathcal{L}(e)} v$, we have that $\chi(e) \le \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(e)}$.*

[9] Whenever the term Δ appears in an amortized bound, this should be interpreted as being an upper bound on the maximum degree across the whole sequence of updates. In the introduction, we replaced the $\log \Delta$ term with $\log n$ for simplicity.

[10] Note that $2\beta(1+\epsilon)^2\alpha = (4 + O(\epsilon))\alpha$.

The following lemma shows that our algorithm can be used to maintain a good edge coloring.

▶ **Lemma 17.** *Let χ be a good edge coloring of the graph G w.r.t. $\mathcal{Z}$ and let $S \subseteq E$ be the set of edges that are left uncolored by χ. Then running* ExtendColoring(S)*:*

1. *changes the colors of at most $L^2|S|$ edges in G, and*
2. *turns χ into a good edge coloring with no uncolored edges.*

Proof. We begin by proving (1). Given some edge f, define the potential of f by

$$\Psi(f) = L(\mathcal{L}(f) - 1) + \ell_{\mathcal{L}(f)}(f).$$

Given the set of edges S, define the potential of S as $\Psi(S) = \sum_{f\in S} \Psi(f)$. By Lemma 14, we have that, for any edge f, $1 \leq \Psi(f) = L(\mathcal{L}(f) - 1) + \ell_{\mathcal{L}(f)}(f) \leq L(L - 1) + L = L^2$. Hence, $|S| \leq \Psi(S) \leq L^2|S|$. During each iteration of the while loop in Algorithm 9, exactly one edge receives a new color (and at most one edge becomes uncolored). We now show that during each iteration of the loop, $\Psi(S)$ drops by at least one, implying that we have at most $L^2|S|$ iterations in total, changing the colors of at most $L^2|S|$ many edges. Let f be the edge in S that we are coloring during some iteration of the loop and let c be the color that it receives. During the iteration, we remove f from S; furthermore, if there exists some edge f' colored with c that shares an endpoint with f, we uncolor f' and place it in S. If there is no such edge f', then $\Psi(S)$ drops by at least 1 since we remove f from S and $\Psi(f) \geq 1$. Suppose that there is such an edge f'. We now argue that $\Psi(f') < \Psi(f)$. We first note that one of the endpoints of f' is not contained in $Z_{i,j}$ where $i = \ell_{\mathcal{L}(f)}(u)$ and $j = \mathcal{L}(f)$. This implies that $\ell_{\mathcal{L}(f)}(f') < \ell_{\mathcal{L}(f)}(f)$, so $\mathcal{L}(f') \leq \mathcal{L}(f)$. Hence, if $\mathcal{L}(f) = \mathcal{L}(f')$, it follows that $\Psi(f') < \Psi(f)$. Otherwise, $\mathcal{L}(f') < \mathcal{L}(f)$, and we have that

$$\Psi(f) - \Psi(f') = L(\mathcal{L}(f) - \mathcal{L}(f')) + \ell_{\mathcal{L}(f)}(f) - \ell_{\mathcal{L}(f')}(f') \geq L + (1 - L) \geq 1.$$

In either case, $\Psi(S)$ drops by at least 1. We now prove (2). Let $f = (u, v)$ be the edge in S that we are coloring during some iteration of the while loop such that $u \prec_{\mathcal{L}(f)} v$. We need to show that the color c picked by the algorithm satisfies $c \leq \deg(v) + 2\beta(1 + \epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$. It will then follow by induction that the coloring produced by calling ExtendColoring(S) is good given that we start with a good coloring. We first note that $|C_v| \leq \deg(v) - 1$. Now note that $|C_u^+| \leq \deg^+_{\mathcal{L}(f)}(u) - 1$. Since $\deg^+_{\mathcal{L}(f)}(u) \leq 2\beta(1 + \epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$, we get the desired bound on c. Finally, note that at the start of each iteration, the uncolored edges correspond to exactly the edges in S. Since the algorithm terminates if and only if $S = \varnothing$ and we know that the algorithm terminates after at most $L^2|S|$ many iterations, it follows that the resulting coloring has no uncolored edges. ◀

▶ **Lemma 18.** *The dynamic algorithm maintains a $(\Delta + 2\beta(1 + \epsilon)^2\alpha)$-edge coloring of the graph.*

Proof. By showing that our algorithm maintains a good edge coloring, it follows by Corollary 15 that, for any edge $e \in E$, we have $\chi(e) \leq \Delta + 2\beta(1 + \epsilon)\tilde{\alpha}_{\mathcal{L}(e)} \leq \Delta + 2(1 + \epsilon)^2\alpha$. We do this by showing that, after an update, the algorithm uncolors all of the edges $f = (u, v)$ in the graph that don't satisfy the condition $\chi(f) \leq \deg(v) + 2\beta(1 + \epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$ for $u \prec_{\mathcal{L}(f)} v$ in the updated decomposition system, places them in a set S, and calls Algorithm 9 on the set S. By Lemma 17, it then follows that the algorithm maintains a good coloring of the entire graph.

We refer to an edge $e = (u, v)$ as *bad* if it does not satisfy the condition required by a good coloring, i.e. if $\chi(e) \neq \perp$ and $\chi(e) > \deg(v) + 2\beta(1 + \epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$ where $u \prec_{\mathcal{L}(f)} v$. Suppose we have a good edge coloring of the entire graph and insert an edge e into the graph. Since this cannot decrease the degrees of any nodes or change the levels of any edges (since we have not yet updated the decomposition system) this cannot cause any edges to become bad. On the other hand, if we delete an edge $e = (u, v)$, some of the edges incident to u and v might become bad since $\deg(u)$ and $\deg(v)$ decrease by 1. Any such edges that become bad must be contained within the set $\Gamma_u \cup \Gamma_v$ where

$$\Gamma_w = \{f = (w', w) \in N(w) \mid w' \prec_{\mathcal{L}(f)} w \text{ and } \chi(f) > \deg(w) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}\}$$

where the degrees are w.r.t. the state of the graph G *after* the deletion of e. If we uncolor all of the edges in $\Gamma_u \cup \Gamma_v$, we restore χ to being a good edge coloring. After updating the decomposition system, the levels of some edges might change in some layers. Any edge that does not change levels in any layer will *not* become bad, since $\mathcal{L}(f)$ (and hence $\tilde{\alpha}_{\mathcal{L}(f)}$) and its orientation in $\prec_{\mathcal{L}(f)}$ do not change. However, an edge f that changes levels in some layer might become bad if $\mathcal{L}(f)$ decreases (causing the value of $\tilde{\alpha}_{\mathcal{L}(f)}$ to decrease) or if its orientation with respect to $\prec_{\mathcal{L}(f)}$ changes. Hence, we uncolor all such edges.[11] This guarantees that there are no bad edges when we call ExtendColoring. Since we give ExtendColoring all of the edges that are uncolored, it follows that we maintain a good edge coloring of the entire graph. ◀

▶ **Lemma 19.** *The dynamic algorithm has $O(\log^4 n/\epsilon^5)$ amortized recourse per update.*

Proof. Suppose that our algorithm handles a sequence of T updates (edge insertions or deletions) starting from an empty graph. Let $S^{(t)}$ denote the set of edges uncolored by our algorithm during the t^{th} update before calling ExtendColoring on the set $S^{(t)}$. By Lemma 17, we know that at most $L^2|S^{(t)}| = O(|S^{(t)}|\log^2 n/\epsilon^2)$ many edges will change color during this update. By showing that $(1/T) \cdot \sum_{t\in[T]} |S^{(t)}|$ is $O(\log^2 n/\epsilon^3)$, our claimed amortized recourse bound follows. Now fix some $t \in [T]$ and let $e = (u, v)$ be the edge being either inserted or deleted during this update. The edges uncolored by the algorithm while handling this update are either contained in the set $\Gamma_u \cup \Gamma_v$ (if the update is a deletion) or change levels in some layer after we update the decomposition system. There can only be at most $2L$ many edges of the former type. This is because, given some $j \in [L]$, there is at most one edge $f \in \Gamma_w$ with $\mathcal{L}(f) = j$ such that $\chi(f) > \deg(w) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$. Otherwise, since all the edges incident on w have distinct colors, there exists such an edge f such that $\chi(f) > \deg(w) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)} + 1$, which contradicts the fact that χ was a good coloring of the graph before the deletion of e. It follows that $|\Gamma_w \cap \mathcal{L}^{-1}(j)| \le 1$, so

$$|\Gamma_w| = \sum_{j\in[L]} |\Gamma_w \cap \mathcal{L}^{-1}(j)| \le L$$

and hence $|\Gamma_u \cup \Gamma_v| \le 2L$. To bound the number of edges that changed levels in at least one of the decompositions in the decomposition system, recall (see Proposition 7) that the amortized recourse of the algorithm that maintains the decomposition system is $O(L^2/\epsilon)$. It follows that the amortized number of such edges is $O(L^2/\epsilon)$. We have that

$$\frac{1}{T} \cdot \sum_{t\in[T]} |S^{(t)}| = O\left(\frac{L^2}{\epsilon}\right) + 2L = O\left(\frac{\log^2 n}{\epsilon^3}\right).$$ ◀

[11] Note that these are precisely the edges that contribute towards the recourse of the dynamic decomposition system.

References

1 Leonid Barenboim, Michael Elkin, and Tzalik Maimon. Deterministic distributed $(\Delta + o(\Delta))$-edge-coloring, and vertex-coloring of graphs with bounded diversity. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 175–184. ACM, 2017.

2 Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland*, volume 108 of *Procedia Computer Science*, pages 89–98. Elsevier, 2017.

3 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 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018.

4 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Density-sensitive algorithms for $(\Delta + 1)$-edge coloring. *CoRR*, abs/2307.02415, 2023. `arXiv:2307.02415`.

5 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 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3393–3440. SIAM, 2024.

6 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182. ACM, 2015.

7 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2633–2652. SIAM, 2018.

8 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

9 Aleksander B. J. Christiansen, Eva Rotenberg, and Juliette Vlieghe. Sparsity-parameterised dynamic edge colouring. *CoRR*, abs/2311.10616, 2023. `arXiv:2311.10616`.

10 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, 2023.

11 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1937–1945. SIAM, 2019.

12 Łukasz Kowalik. Edge-coloring sparse graphs with Δ colors in quasilinear time. *arXiv preprint*, 2024. `arXiv:2401.13839`.

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