Nibbling at Long Cycles: Dynamic (and Static) Edge Coloring in Optimal Time

Sayan Bhattacharya[1] Martín Costa[1] Nadav Panski[2] Shay Solomon[2]

[1]University of Warwick
[2]Tel Aviv University

Abstract

We consider the problem of maintaining a $(1+\epsilon)\Delta$-edge coloring in a dynamic graph G with n nodes and maximum degree at most Δ. The state-of-the-art update time is $O_\epsilon(\text{polylog}(n))$, by Duan, He and Zhang [SODA'19] and by Christiansen [STOC'23], and more precisely $O(\log^7 n/\epsilon^2)$, where $\Delta = \Omega(\log^2 n/\epsilon^2)$.

The following natural question arises: What is the best possible update time of an algorithm for this task? More specifically, **can we bring it all the way down to some constant** (for constant ϵ)? This question coincides with the *static* time barrier for the problem: Even for $(2\Delta-1)$-coloring, there is only a naive $O(m \log \Delta)$-time algorithm.

We answer this fundamental question in the affirmative, by presenting a dynamic $(1+\epsilon)\Delta$-edge coloring algorithm with $O(\log^4(1/\epsilon)/\epsilon^9)$ update time, provided $\Delta = \Omega_\epsilon(\text{polylog}(n))$. As a corollary, we also get the first linear time (for constant ϵ) *static* algorithm for $(1+\epsilon)\Delta$-edge coloring; in particular, we achieve a running time of $O(m \log(1/\epsilon)/\epsilon^2)$.

We obtain our results by carefully combining a variant of the Nibble algorithm from Bhattacharya, Grandoni and Wajc [SODA'21] with the subsampling technique of Kulkarni, Liu, Sah, Sawhney and Tarnawski [STOC'22].

Contents

1 Introduction

Given an n-node graph $G = (V, E)$ and a *palette* of *colors* $\mathcal{C}$, a *(proper) edge coloring* $\chi : E \to \mathcal{C}$ assigns a color to each edge of G while ensuring that no two neighboring edges receive the same color. If $|\mathcal{C}| = \lambda$, then we say that χ is a λ-edge coloring of G. It is easy to observe that we need at least Δ colors for any proper edge coloring, where Δ is the maximum degree in the input graph, and a textbook theorem by Vizing [Viz64] guarantees that any graph admits a $(\Delta+1)$-edge coloring. In contrast, a simple greedy algorithm, which scans the edges of G in any arbitrary order and assigns a "free" color to each edge during the scan, returns a $(2\Delta - 1)$-edge coloring of G.

In this paper, we focus on the problem of maintaining an edge coloring in a dynamic setting. Here, the input graph G undergoes a sequence of *updates* (edge insertions/deletions), but its maximum degree always remains at most a known parameter Δ. We wish to maintain a λ-edge coloring in this dynamic graph G, for as small a value of λ as possible. We also wish to ensure that the *update time* of our algorithm, which is the time it takes to handle an update in G, remains small. Thus, the key challenge is to understand the trade off between the number of colors needed by an algorithm and its update time. We now summarize the state-of-the-art for this problem.

We can maintain a $(2\Delta - 1)$-edge coloring in $O(\log \Delta)$ update time [BM17, BCHN18], which essentially requires dynamizing the greedy algorithm using a variant of binary search. If we wish to move beyond the greedy threshold of $(2\Delta - 1)$, then there are two further results. We know how to maintain a $(1+\epsilon)\Delta$-edge coloring in $O(\log^7 n/\epsilon^2)$ update time when $\Delta = \Omega(\log^2 n/\epsilon^2)$ [DHZ19], and a $(1+\epsilon)\Delta$-edge coloring in $O(\log^9 n \log^6 \Delta/\epsilon^6)$ update time with no restrictions on Δ [Chr23]. Given that both these two results incur a *large* polylogarithmic factor in their update times, it is very natural to ask the following question, which we address in this paper.

Consider any arbitrarily small constant $\epsilon \in (0, 1)$, and suppose that we wish to maintain a $(1+\epsilon)\Delta$-edge coloring in a dynamic graph. Then what is the best possible update time of any dynamic algorithm for this task? Can we bring this update time down all the way to $O(1)$?

As we will shortly see, this paper answers the above question in the affirmative. Before stating our formal result, we outline the major obstacles that we need to overcome to achieve this goal.

1.1 Perspective: The Quest for Constant Update Time

Achieving constant update times for fundamental problems is an important research agenda within dynamic algorithms [AS21, BCH17, BGK$^+$22, BGM17, BHNW21, BK19, HP20, PS16, Sol16, SW18]. There are two major considerations that underpin this research agenda. (i) A constant update time algorithm rules out the possibility of obtaining a (cell-probe) lower bound for the concerned problem [Lar12, PD06]. (ii) It immediately implies a *linear time* algorithm for the concerned problem in the static setting,[1] and thus aligns with what is essentially the best possible static guarantee. With this backdrop, we encounter a significant hurdle at the very beginning of our quest, since currently there does not even exist a $O_\epsilon(m)$ time *static* algorithm for $(1+\epsilon)\Delta$-edge coloring.[2] To elaborate on this further, we now review the state-of-the-art for static edge coloring algorithms.

The greedy algorithm can easily be implemented by means of a binary search, which gives us $(2\Delta - 1)$-edge coloring in $O(m \log \Delta)$ time, where m is the number of edges in the input graph. Beyond the greedy threshold, it is known how to compute a $(\Delta + 1)$-edge coloring in $O(m\sqrt{n})$

[1] We take the (static) input graph, and feed it to the dynamic algorithm by inserting its edges one at a time.

[2] Throughout this paper, the notation $O_\epsilon(.)$ hides poly$(1/\epsilon)$ factors.

time [GNK+85, Sin19], and a $\left(\Delta + \Delta^{0.5+\epsilon}\right)$-edge coloring in $\tilde{O}_\epsilon(m)$ time.[3] Finally, the two dynamic algorithms [Chr23, DHZ19] immediately imply static algorithms for $(1+\epsilon)\Delta$-edge coloring, with running times $O(m \log^7 n/\epsilon^2)$ and $O(m \log^9 n \log^6 \Delta/\epsilon^6)$ respectively. In addition, [DHZ19] also obtain a $(1+\epsilon)\Delta$-edge coloring algorithm with $O(m \log^6 n/\epsilon^2)$ running time, when $\Delta = \Omega(\log n/\epsilon)$.

In fact, if we insist upon getting an exact linear (i.e., $O(m)$) running time, and subject to this constraint try to minimize the number of colors being used, then the only game in town happens to be a very simple, folklore algorithm that gives us $(2+\epsilon)\Delta$-coloring. This algorithm can also be dynamized to get $O(1/\epsilon)$ update time, as explained below.

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

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

(I) The two known dynamic algorithms for $(1+\epsilon)\Delta$-edge coloring [DHZ19, Chr23] are both analyzed in a *memory-less* manner. Specifically, they assume that we start with any arbitrary, adversarially chosen $(1+\epsilon)\Delta$-edge coloring in the current graph G, and then show how to modify that coloring (to ensure that it remains proper) in polylogarithmic time after the insertion/deletion of an edge. A lower bound construction from [CHL+18], however, implies that any such memory-less analysis must necessarily imply an update time of $\Omega(\log(\epsilon n)/\epsilon)$.

(II) There is a weaker version of the dynamic edge coloring problem, where we care about the *recourse* (as opposed to update time) of the maintained solution, which basically equals the number of changes the algorithm makes to the coloring after an update. There exists a $(1+\epsilon)\Delta$-edge coloring algorithm with $O_\epsilon(1)$ recourse [BGW21], based on the Nibble method (see Section 1.2 for more details) that was first used in the context of edge coloring in the distributed setting [DGP98]. It seems very difficult to implement the algorithm of [BGW21] using $O_\epsilon(1)$ update time data structures, for two reasons. First, the [BGW21] dynamic algorithm needs to resample the color c of an edge $e = (u, v)$ when its palette $P(e) = \mathcal{C} \setminus \left(\overline{P(u)} \cup \overline{P(v)}\right)$ changes by a small amount, even if c continues to be part of $P(e)$. It is not at all clear how to implement this resampling efficiently, i.e., in constant update time. Second, and more fundamentally, all existing Nibble method-based algorithms require some form of *regularization gadget*, since the inductive approach that is used to analyze these algorithms does not work on graphs that are not near-regular. Implementing this gadget requires $\Omega(n\Delta)$ running time in the static setting, and $\Omega(n\Delta)$ preprocessing time in the dynamic setting.

[3]The $\tilde{O}(.)$ notation hides polylog(n) factors.

1.2 Our Results

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

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

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

Theorem 1.1. *In the static setting, we can compute a $(1+\epsilon)\Delta$-edge coloring in the input graph G in $O(m\log(1/\epsilon)/\epsilon^2)$ time w.h.p., provided $\Delta \geq \Delta^\star$.*

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

Theorem 1.2. *We can maintain a $(1+\epsilon)\Delta$-edge coloring in a dynamic graph G in $O(\log^4(1/\epsilon)/\epsilon^9)$ expected worst-case update time (against an oblivious adversary), provided $\Delta \geq \Delta^\star$.*

We can convert the expected worst-case update time bound of Theorem 1.2 into a high probability amortized update time guarantee, for polynomially long update sequences (see Corollary C.6).

1.3 Our Technique

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

Our static algorithm. We start by summarizing the Nibble algorithm (see Section 2.1). It runs in $T := \lfloor(1/\epsilon)\log(1/\epsilon)\rfloor$ rounds, on the input graph $G = (V, E)$ with a palette $\mathcal{C}$ of $(1+\epsilon)\Delta$ colors. At the start of round $i \in [T]$, let $P_i(v)$ denote the palette of a node $v \in V$, which consists of all the colors that have *not* yet been (tentatively) assigned to any edge incident on v. In round i, each uncolored edge e *selects* itself independently with probability ϵ. Next, every selected edge $e = (u, v)$ picks a tentative color $\tilde{\chi}(e)$ independently and u.a.r. from its palette $P_i(u) \cap P_i(v)$. At the end of T rounds, we collect all the *failed* edges $F \subseteq E$; these are the edges that were either not selected during any of the T rounds and hence did not receive any tentative color, or received a tentative color which conflicts with one of its neighbors. We now color the subgraph $G_F := (V, F)$, using the folklore algorithm and an extra palette of $O(\Delta(G_F))$ colors that is mutually disjoint with $\mathcal{C}$. This, combined with the tentative colors assigned to the edges in $E \setminus F$, gives us a proper $(1+\epsilon)\Delta + O(\Delta(G_F))$-coloring of G. The main challenge now is to show that $\Delta(G_F) = O(\epsilon\Delta)$ w.h.p., for that would give us a $(1+O(\epsilon))\Delta$-coloring of G.

Next, we observe that we do not need any regularizing gadget to analyze the Nibble algorithm, *if the input graph G is a forest* (see Section 2.2). This is primarily because under such a scenario, just before we pick a tentative color for an edge $(u, v) \in E$ in some round $i \in [T]$, the palettes $P_i(u)$ and $P_i(v)$ are mutually independent. This observation makes it easy to obtain a $O_\epsilon(m)$ time implementation of the Nibble algorithm for $(1+\epsilon)\Delta$-edge coloring. We essentially use the same hash table data structures which allow us to efficiently implement the folklore algorithm (see Section 1.1), along with the fact that w.h.p., at the start of each round $i \in [T]$, the palette $P_i(u, v) := P_i(u) \cap P_i(v)$ of each uncolored edge is of size $\Omega(\epsilon^2\Delta)$ (see Corollary 2.8).

At this point, we move on to the general case where the input graph G might contain cycles (see Section 2.3). Here, we first observe that the palette $P_i(v)$ of a node v for a round $i \in [T]$ depends only on the i-hop neighborhood of v. Say that a node v is *good* in G if its $(T+1)$-hop neighborhood in G does not contain any cycle, and *bad* otherwise. Since the Nibble algorithm runs for only T rounds, we can simply pretend that the input graph is a forest while analyzing what the algorithm does to a good node and all its incident edges (see Lemma 2.17). In particular, we can show that w.h.p. every good node will have degree at most $O(\epsilon\Delta)$ in G_F.

We now combine the previous observations with a subsampling technique [KLS+22] (see Section 2.4). The basic idea is simple. Fix two parameters $\gamma := 1/(30T)$ and $\Delta' := \Delta^\gamma$, and set $\eta := \Delta/\Delta'$. Next, partition the input graph G into η subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$, by placing each edge $e \in E$ independently and u.a.r. in one of the subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$. This partition happens to satisfy the following two properties. (I) W.h.p. $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta'$ for all $j \in [\eta]$. (II) Say that an edge $e = (u, v) \in E$ is *problematic* iff either u or v is a bad node in $\mathcal{G}_j$, where $j \in [\eta]$ is the unique index such that $e \in \mathcal{G}_j$. Let $E^\star \subseteq E$ be the set of all problematic edges, and let $G^\star := (V, E^\star)$. Then $\Delta(G^\star) = O(\epsilon\Delta)$ w.h.p. Armed with these two observations, our final algorithm on general graphs works as follows. We compute the partition of the input graph G into the subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$. For each $j \in [\eta]$, we run the Nibble algorithm on $\mathcal{G}_j$ in an attempt to color it with a (distinct) palette $\mathcal{C}_j$ of $(1 + O(\epsilon))\Delta'$ colors. Overall, this requires $\eta \cdot (1 + O(\epsilon))\Delta' = (1 + O(\epsilon))\Delta$ colors. At the end of this step, we are left with two types of failed edges that could not be properly colored: the ones that are problematic, and the ones that are not. Because of the locality of the Nibble method, for each $j \in [\eta]$, w.h.p. each node $v \in V$ is incident on at most $O(\epsilon\Delta')$ non-problematic failed edges in $\mathcal{G}_j$. Thus, w.h.p. the maximum degree of the subgraph consisting of all non-problematic edges over all $j \in [\eta]$ is at most $\eta \cdot O(\epsilon\Delta') = O(\epsilon\Delta)$. Next, by Property II above, the maximum degree of the subgraph consisting of all problematic edges, over all $j \in [\eta]$, is given by $\Delta(G^\star) = O(\epsilon\Delta)$. Putting everything together, we infer that at the end of the first step, the subgraph consisting of all failed edges has maximum degree $O(\epsilon\Delta)$. Hence, we can easily color these remaining failed edges, using the folklore algorithm from Section 1.1 and an extra set of $O(\epsilon\Delta)$ colors. This leads to a $(1 + O(\epsilon))\Delta$-coloring of the input graph G, without any additional overhead in the running time compared to the scenario where G was a forest, because the subsampling step can easily be implemented very efficiently.

Our dynamic algorithm. We dynamize our static algorithm using a very natural approach (see Section 3). It is not surprising that the subsampling step is relatively easy to dynamize, so here we only focus on highlighting what our dynamic algorithm does when the input graph remains a forest. Essentially, our dynamic algorithm hinges upon two main observations.

(I) When an edge e gets inserted, we might as well fix an index $i_e \in [T+1]$ by sampling i_e from a capped geometric distribution with probability ϵ,[4] and we can also fix an infinite length color-sequence c_e, such that for each $\ell \in \mathbb{Z}^+$ the ℓ^{th} entry $c_e(\ell)$ in this sequence is a color sampled independently and u.a.r. from the input palette $\mathcal{C}$. These rounds $\{i_e\}_e$ and color-sequences $\{c_e\}_e$ uniquely determine the output of the Nibble algorithm on a given input graph $G = (V, E)$. Specifically, each edge $e \in E$ selects itself in round i_e, and then identifies the smallest integer $\ell \in \mathbb{Z}^+$ such that $c_e(\ell) \in P_i(e)$, and sets $\tilde{\chi}(e) := c_e(\ell)$.[5] Throughout the sequence of updates, we simply maintain the output of this static algorithm w.r.t. the indices $\{i_e\}_e$ and color-sequences $\{c_e\}_e$. We show that this natural approach itself suffices to guarantee an expected worst-case recourse of $O_\epsilon(1)$ (see Section 3.1), and is in sharp contrast with the algorithm of [BGW21] which required repeated resampling of colors.

(II) We need one additional insight to implement our low-recourse algorithm in $O_\epsilon(1)$ update time. Specifically, we *truncate* the color-sequences $\{c_e\}$ at length $K := \Theta((1/\epsilon^2)\log(1/\epsilon))$, i.e., for each

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

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

edge e, we stop constructing the sequence c_e after sampling the first K colors $c_e(1), \ldots, c_e(K)$. The intuition behind why everything still works is as follows. W.h.p., we know that $|P_{i_e}(e)| = \Omega(\epsilon^2 \Delta)$. Thus, conditioned on this event, at least one of the first K colors in c_e should appear in $P_i(e)$ with probability at least $1 - \epsilon$. In other words, the resulting algorithm is equivalent to the following process: We throw away each edge $e \in E$ with probability ϵ, and we run the NIBBLE algorithm on the surviving edges. The subgraph consisting of the edges that get thrown away has maximum degree $O(\epsilon\Delta)$ w.h.p., and hence we can separately color this subgraph using the folklore algorithm from Section 1.1. It turns out that we can develop data structures that support the implementation of this modified algorithm in $O_\epsilon(1)$ expected worst-case update time (see Section 3.2). The main reason is that the truncation step acts in a way which is reminiscent of *palette-sparsification* [DHZ19]. Indeed, now each edge gets assigned a tentative color that comes from a small set of size $K = O_\epsilon(1)$. This helps us design a supporting data structure whose expected update time is proportional to the recourse of the algorithm from step (I) above, which, we already know to be $O_\epsilon(1)$.

1.4 Remark on the Lower Bound on Δ

In the edge coloring literature, it is common to assume that $\Delta = \Omega(\text{polylog}(n))$ [BGW21, DHZ19, KLS+22]. In this paper, the reason we need the lower bound on Δ is as follows (see Section 2.4 for details). We partition the input graph G into $\eta = \Delta/\Delta'$ subgraphs $G_1, \ldots, G_\eta$, by throwing each edge of G u.a.r. into one of these subgraphs. We now need to enforce the following two properties. (i) Δ' needs to be large enough to ensure that we have enough concentration to be able to run the Nibble algorithm on each G_i. (ii) Δ' needs to be small enough to ensure that each G_i is sufficiently "locally treelike" for our analysis to go through. In particular, so that the subgraph $G^\star$, which consists of all the bad edges, has maximum degree at most $\epsilon\Delta$ (see Claim 2.19), because these bad edges will get separately colored using a greedy algorithm. Now, it so happens that for Property (i) to hold, we need $\Delta' \geq \Omega(\log n/\epsilon^4)$, and for Property (ii) to hold, we need $\Delta' \leq \Delta^{\Theta(1/T)}$, where $T = \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ is the number of rounds required by the Nibble method. Combining these two inequalities together, we get that $\Delta \geq \left(\log n/\epsilon^4\right)^{\Theta((1/\epsilon)\log(1/\epsilon))}$. We leave it as a challenging open question to improve this lower bound on Δ.

2 Overview of our Static Algorithm

In this section, we describe how our algorithm (see Theorem 1.1) works in the static setting, and present an overview of the key ideas that underpin its analysis. To convey the main intuition behind our framework, here we intentionally explain some of the arguments in an informal/semi-rigorous manner. The complete, formal proofs from this section are deferred to Appendix A and Appendix D.

Organization. In Section 2.1, we present the NIBBLE algorithm. Section 2.2 explains how to analyze and implement this algorithm when the input graph is a forest. In Section 2.3, we show that on general graphs, the analysis from Section 2.2 still holds for all those nodes that are *not* part of any short cycle. Finally, Section 2.4 contains an overview of our final algorithm, which involves combining the NIBBLE method along with a subsampling technique of [KLS+22].

2.1 The NIBBLE Algorithm

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

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

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

Observation 2.1. *$\tilde{\chi}$ is a proper $(1 + \epsilon)\Delta$-edge coloring in the subgraph $G_{E \setminus F} := (V, E \setminus F)$.*

The pseudocode of this procedure appears in Algorithm 1. We now describe some further notations that will be used throughout the rest of this paper.

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

Intuitively, it is easy to see that the NIBBLE algorithm is *symmetric* w.r.t. the palette $\mathcal{C}$, i.e., it does not give preference to one color over another. The lemma below formalizes this intuition, and will be repeatedly invoked during our analysis (we defer its proof to Appendix A.2.2).

Lemma 2.2. *Fix the random bits used by the* NIBBLE *algorithm that determine which edges get selected in which rounds. Then for all $u \in V$, $i \in [T]$, $C \subseteq \mathcal{C}$ and permutations $\pi : \mathcal{C} \longrightarrow \mathcal{C}$, we have*

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

2.2 Analysis of the NIBBLE Algorithm on Forests

The NIBBLE algorithm returns a proper $(1+\epsilon)\Delta$-edge coloring $\tilde{\chi}$ on the subgraph $G_{E \setminus F} = (V, E \setminus F)$ (see Observation 2.1). We now wish to argue that the remaining subgraph $G_F := (V, F)$ has small maximum degree. Specifically, if it so happens that $\Delta(G_F) = O(\epsilon\Delta)$, then we can just color the edges of G_F, using the folklore algorithm from Section 1.1 and an extra set of $O(\epsilon\Delta)$ colors. This, combined with the coloring $\tilde{\chi}$ of $G_{E \setminus F}$, would give us a $(1 + O(\epsilon))\Delta$-coloring of G. We now prove this desired upper bound on $\Delta(G_F)$, under the assumption that the input graph G is a forest with sufficiently large maximum degree. The main result in this section is summarized below.

Algorithm 1 $\textsc{Nibble}(G = (V, E), \Delta, \epsilon)$

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

Lemma 2.3. *Let G be a forest with maximum degree $\Delta \geq \frac{(100 \log n)}{\epsilon^4}$. Then w.h.p. we have:*

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

We start with the following key observation.

Observation 2.4. *Let $G = (V, E)$ be a forest. Fix the random bits used by the* Nibble *algorithm that determine which edges get selected in which rounds. Consider any $i \in [T]$, $u \in V$, and edges $(u, v_1), \ldots, (u, v_k) \in S_i$. Then the palettes $\{P_i(u), P_i(v_1), \ldots, P_i(v_k)\}$ are mutually independent.*

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

In Lemma 2.5, Corollary 2.6 and Lemma 2.7, we derive concentration bounds on the sizes of the sets $N_i(v)$ and the palettes of nodes/edges at the start of each round. Later on, we use these concentration bounds in Section 2.2.1, which gives an overview of the proof of Lemma 2.3. Finally, we explain how to efficiently implement the Nibble algorithm in linear time in Section 2.2.2.

Lemma 2.5. *Let $\mathcal{Z}$ denote the event which occurs iff we have $|N_i(u)| < (1+\epsilon) \cdot \epsilon(1-\epsilon)^{i-1}\Delta$ for all nodes $u \in V$ and all rounds $i \in [T]$. Then the event $\mathcal{Z}$ occurs w.h.p.*

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

Corollary 2.6. *Conditioned on $\mathcal{Z}$, we always have $|P_i(u)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta$ for all $u \in V, i \in [T]$.*

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

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

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

Lemma 2.7. *Fix the random bits used by the* NIBBLE *algorithm that determine which edges are selected in which rounds, in* any *way which ensures that the event $\mathcal{Z}$ occurs. Then w.h.p., we have $|P_i(e)| > (1-\epsilon^2)(1-\epsilon)^{2(i-1)}\Delta$ for all rounds $i \in [T]$ and all edges $e \in E_i$.*

Proof. (Sketch) Consider any $i \in [T]$ and any edge $e = (u, v) \in E_i$. For each color $c \in \mathcal{C}$ and each node $w \in V$, let $X_c^w \in \{0, 1\}$ be an indicator random variable that is set to 1 iff $c \in P_i(w)$. Clearly, we have $|P_i(e)| = \sum_{c \in \mathcal{C}} X_c^u \cdot X_c^v$. By Observation 2.4, X_c^u and X_c^v are independent, and hence:

$$\mathbb{E}\left[|P_i(e)|\right] = \sum_{c \in \mathcal{C}} \mathbb{E}\left[X_c^u \cdot X_c^v\right] = \sum_{c \in \mathcal{C}} \mathbb{E}\left[X_c^u\right] \cdot \mathbb{E}\left[X_c^v\right]. \tag{2}$$

We now consider any fixed color $c \in \mathcal{C}$, and derive a lower bound on $\mathbb{E}[X_c^u]$.[6] Using the symmetry of the NIBBLE algorithm w.r.t. the colors (see Lemma 2.2), in conjunction with the fact that $|P_i(u)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta$ (see Lemma 2.5), we get:

$$\mathbb{E}\left[X_c^u\right] = \Pr\left[X_c^u = 1\right] \geq \frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{|\mathcal{C}|}. \tag{3}$$

Since $|\mathcal{C}| = (1+\epsilon)\Delta$, from Equation (2) and Equation (3), we derive that:

$$\mathbb{E}\left[|P_i(e)|\right] \geq |\mathcal{C}| \cdot \left(\frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{|\mathcal{C}|}\right)^2 = (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta. \tag{4}$$

With some extra effort, we can modify the above argument by working with a slightly different (but analogous) set of random variables Y_c^w (instead of X_c^w) such that the collection of random variables $\{Y_c^u \cdot Y_c^v\}_{c \in \mathcal{C}}$ is *negatively associated* (see Definition E.5). This allows us to derive a concentration bound out of Equation (4), which leads to the proof of the lemma (see Appendix A for details). □

Corollary 2.8. *Fix the random bits used by the* NIBBLE *algorithm that determine which edges are selected in which rounds, in such a way that the event $\mathcal{Z}$ occurs. Then w.h.p., we have $|P_i(e)| \geq \epsilon^2(1+\epsilon)\Delta/8$ for all rounds $i \in [T]$ and all edges $e \in E_i$.*

Proof. (Sketch) Follows from Lemma 2.7 and the observation that $i \leq T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$. □

Corollary 2.8 will be useful in analyzing the runtime of the NIBBLE algorithm in Section 2.2.2.

2.2.1 Proof (Sketch) of Lemma 2.3

Consider any node $v \in V$, and let $N_F(v) = N(v) \cap F$ be the set of *failed edges* incident on v. We will show that $\deg_F(v) = |N_F(v)| = O(\epsilon\Delta)$ w.h.p. Lemma 2.3 will then follow from a union bound over all nodes $v \in V$. We begin by partitioning the set $N_F(v)$ into three subsets:

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

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

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

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

Claim 2.9. *We have* $\left|N_F^{(0)}(v)\right| = O(\epsilon\Delta)$ *w.h.p.*

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

For the rest of Section 2.2.1, we fix the random bits used by the NIBBLE algorithm that determine which edges are selected in which rounds, in such a way that the event $\mathcal{Z}$ occurs (the event $\mathcal{Z}$ occurs w.h.p., according to Lemma 2.5). Lemma 2.3 will follow from Claim 2.9, Claim 2.10 and Corollary 2.13.

Claim 2.10. *We have* $\left|N_F^{(1)}(v)\right| = 0$ *w.h.p.*

Proof. (Sketch) Lemma 2.7 implies that w.h.p. the following event occurs: $P_i(e) \neq \varnothing$ for all $i \in [T]$ and all $e \in S_i$. Conditioned on this event, no edge gets added to the set $N_F^{(1)}(v)$. □

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

Claim 2.11. *Consider any round* $i \in [T]$*, any edge* $(u, v) \in S_i$*, and any endpoint* $x \in \{u, v\}$*. Then we have:* $\Pr[\, \exists\, e \in N_i(x) \setminus \{(u, v)\} : \tilde{\chi}(u, v) = \tilde{\chi}(e)] \leq \epsilon$.

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

Observation 2.4 implies that the palettes $\{P_i(u), P_i(v), P_i(w)\}$ are mutually independent. Furthermore, since we have conditioned on the event $\mathcal{Z}$, Corollary 2.6 lower bounds the sizes of each of these palettes $\{P_i(u), P_i(v), P_i(w)\}$. Now, fix (i.e., condition on) the palettes $P_i(u)$ and $P_i(v)$, and let $c = \tilde{\chi}(u, v) \in P_i(u) \cap P_i(v)$ be the tentative color received by the edge (u, v) in round i. Using the symmetry of the NIBBLE algorithm w.r.t. the colors (see Lemma 2.2), and recalling that $|P_i(u)| \geq (1+\epsilon)(1-\epsilon)^{i-1}\Delta$ as per Corollary 2.6, we get:

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

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

$$\Pr\left[\tilde{\chi}(u, v) \neq \tilde{\chi}(u, w) \;\forall\; (u, w) \in N_i(u) \setminus \{(u, v)\}\right] \geq \left(1 - \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}\right)^{|N_i(u)|} \geq 1 - \epsilon. \quad (5)$$

The first inequality holds because of Observation 2.4, whereas the last inequality holds since we have conditioned on the event $\mathcal{Z}$ (see Lemma 2.5). The lemma follows if we consider the probability of the complement of the event captured by Equation (5). □

Claim 2.12. *Consider any round $i \in [T]$. Then we have: $\left|N_F^{(2)}(v) \cap S_i\right| = O(\epsilon^2(1-\epsilon)^{i-1}\Delta)$ w.h.p.*

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

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

Observe that $N_F^{(2)}(v) \subseteq F_i'(v) \cup F_i''(v)$. Now, consider any edge $(u,v) \in N_i(v)$. By Claim 2.11, we have $\Pr[(u,v) \in F_i'(v)] \le \epsilon$ and $\Pr[(u,v) \in F_i''(v)] \le \epsilon$. Thus, by linearity of expectation, we get:

$$\mathbb{E}\left[\left|N_F^{(2)}(v)\right|\right] \le \mathbb{E}\left[\left|F_i'(v)\right|\right] + \mathbb{E}\left[\left|F_i''(v)\right|\right] \le 2\epsilon \cdot |N_i(v)| = O(\epsilon^2(1-\epsilon)^{i-1}\Delta). \tag{6}$$

In the above derivation, the last step holds because we conditioned on the event $\mathcal{Z}$ (see Lemma 2.5). With a little bit of extra work, we can infer that: (a) $|F_i'(v)|$ can be expressed as a function of a collection of random variables that is Lipschitz with all constants 2 (see Definition E.3), which allows us to derive a concentration bound on $|F_i'(v)|$ using the method of bounded differences (see Proposition E.4). (b) Using the fact that the input graph G is a forest, $|F_i''(v)|$ can be expressed as the sum of mutually independent 0/1 random variables, which allows us to derive a concentration bound on $|F_i''(v)|$ by applying a Chernoff bound. This concludes the proof of the claim. □

Corollary 2.13. *We have: $\left|N_F^{(2)}(v)\right| = O(\epsilon\Delta)$ w.h.p.*

Proof. Since $N_F^{(2)}(v) \subseteq \bigcup_{i=1}^{T} S_i$, from Claim 2.12 we derive that whp:

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

□

Lemma 2.3 now follows from Claim 2.9, Claim 2.10, Lemma 2.5 and Corollary 2.13.

2.2.2 Running Time of the Nibble Algorithm on Forests

We now briefly describe how we can implement the Nibble algorithm in linear time when the input graph $G = (V, E)$ is a forest. We begin by scanning through all of the edges $e \in E$, and assigning a round $i_e \in [T+1]$ to each edge e, sampled i.i.d. from a capped geometric distribution with success probability ϵ. We then construct the sets $S_1, \dots, S_{T+1}$, where S_i consists of the edges that will be colored during round i. This can be done in $O(m)$ time. For each $i = 1, \dots, T$, we then scan through all of the edges $e \in S_i$ and sample a color $\tilde{\chi}(e) \sim P_i(e)$ independently and u.a.r. In order to implement this sampling, suppose that we can check whether or not a given color c is contained in $P_i(e)$ in $O(1)$ time. Since we know by Corollary 2.8 that $|P_i(e)| = \Omega(\epsilon^2\Delta)$ w.h.p., we can sample a color c from $\mathcal{C}$ independently and u.a.r. and this color will be contained in $P_i(e)$ with probability $\Omega(\epsilon^2)$. Hence, in expectation, we need to sample $O(1/\epsilon^2)$ many colors before finding one that is contained in $P_i(e)$. It follows that the total expected time required to sample all of the tentative colors is $O(m/\epsilon^2)$. Finally, we can color all the failed edges using the folklore algorithm from Section 1.1 in $O(m)$ time.

It turns out that using hash tables we can maintain, for each node $v \in V$, the complement of its palette, given by $\overline{P(v)} := \{c \in \mathcal{C} : \exists (u,v) \in E \text{ s.t. } \chi(u,v) = c\}$. This allows us to check whether a color is in the palette of an edge $e = (u,v)$ in $O(1)$ time. Specifically, suppose that at the start of round i, we maintain $\overline{P_i(v)}$ for each node v. We can then use this data structure to implement the sampling efficiently as described above. After sampling all of the tentative colors for the edges in S_i, we can then update each of these sets in $O(|S_i|)$ time, by inserting the appropriate colors to each set $\overline{P(v)}$, obtaining $\overline{P_{i+1}(v)}$ for all nodes v. We can also implement data structures that keep track of all the failed edges. We defer all the details of these data structures to Appendix D.

To summarize, we get the following result.

Lemma 2.14. *Given a forest G with maximum degree $\Delta \geq (100 \log n)/\epsilon^4$ as input, the* Nibble *algorithm runs in $O(m/\epsilon^2)$ expected time.*

2.3 Analyzing the Nibble Algorithm on General Graphs

In this section, we analyze the Nibble algorithm when the input graph G may contain cycles. Looking back at the analysis in Section 2.2, it is easy to check that we crucially needed G to be acyclic because we wanted to apply Observation 2.4 in our analysis. This observation was used, for example, in the proof of Lemma 2.7 and in Section 2.2.1. The key insight that we employ in this section is that even if G contains cycles, the preceding claim still holds for all the nodes/edges that are *not* part of any cycle of length $\leq T+1$ (the number of rounds in the Nibble algorithm). Formally, by doing induction on i, we can prove the following lemma.

Lemma 2.15. *Let $\mathcal{N}_G(u,j) := G\left[\{v \in V : dist_G(u,v) \leq j\}\right]$ denote the j-hop neighborhood of any node $u \in V$. Then for every $i \in [T]$, the palette $P_i(u)$ depends only on: $\mathcal{N}_G(u, i-1)$, and the random bits used by the* Nibble *algorithm to implement rounds $j \in \{1, \ldots, i-1\}$ in $\mathcal{N}_G(u, i-1)$.*[7]

We now introduce a key definition below.

Definition 2.16. *We say that a node $v \in V$ is* good *w.r.t. $G = (V,E)$ iff $\mathcal{N}_G(u, T+1)$, the subgraph induced by its $(T+1)$-hop neighborhood in G, does not contain any cycle. Let $U_G \subseteq V$ denote the set of all good nodes in G. We refer to the rest of nodes $B_G := V \setminus U_G$ as* bad *w.r.t. G.*

Informally, since Algorithm 1 only runs for T rounds, and the decisions it makes are *local*, what the algorithm does to some $H \subseteq G$ should only depend on the subgraph of G that is reachable from H by paths of length at most T. Hence, if the $(T+1)$-neighborhood of a node u is acyclic, then it is safe to pretend that the input graph is a forest while analyzing what the algorithm does to the edges incident on u. This implies that even if G contains cycles, we can recover the guarantee of Lemma 2.3 for all the good nodes, which leads us to the following lemma.

Lemma 2.17. *Suppose that the graph G has maximum degree $\Delta \geq \frac{(100 \log n)}{\epsilon^4}$. Then w.h.p. we have:*

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

2.4 The Final (Static) Algorithm: Nibble on Subsampled Graphs

Throughout Section 2.4, we use two parameters: $\gamma := 1/(30T)$ and $\Delta' := \Delta^\gamma$. We also assume that:

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

We now combine Algorithm 1 with the subsampling technique of [KLS+22], which leads to our final algorithm on general graphs. This consists of two steps.

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

Step I: Set $\eta := \Delta/\Delta'$, and partition the input graph $G = (V, E)$ into η subgraphs $\mathcal{G}_1 = (V, \mathcal{E}_1), \ldots, \mathcal{G}_\eta = (V, \mathcal{E}_\eta)$, by placing each edge $e \in E$ uniformly and independently at random into one of the subsets $\mathcal{E}_1, \ldots, \mathcal{E}_\eta$. To ease notation, for all $j \in [\eta]$ we let $\mathcal{U}_j := U_{\mathcal{G}_j}$ and $\mathcal{B}_j := B_{\mathcal{G}_j}$ denote the sets of good and bad nodes in the subgraph $\mathcal{G}_j$ (see Definition 2.16). We now derive two important properties of this subsampling step that will be crucially used in Step II.

Claim 2.18. *W.h.p., for every $j \in [\eta]$ we have $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta'$.*

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

Claim 2.19. *Consider the graph $G^\star = (V, E^\star)$, where $E^\star := \bigcup_{j=1}^{\eta} \{(u,v) \in \mathcal{E}_j : \{u,v\} \cap \mathcal{B}_j \neq \varnothing\}$ is the set of all edges that are incident on bad nodes in their corresponding subsampled graphs. Then w.h.p., we have $\Delta(G^\star) = o(\Delta)$.*

We defer the proof of Claim 2.19 to Section 2.4.1, and proceed to the next step of our algorithm.

Step II: For each $j \in [\eta]$, taking Claim 2.18 into account, we invoke Algorithm 1 on $\mathcal{G}_j$ with a palette $\mathcal{C}_j$ (i.e., the set $\mathcal{C}$ in Line 1 of Algorithm 1) of $(1+\epsilon)^2\Delta'$ colors. The palettes $\mathcal{C}_1, \ldots, \mathcal{C}_j$ are mutually disjoint. For each $j \in [\eta]$, let $\mathcal{F}_j$ be the set of *failed edges* at the end of the corresponding invocation of the NIBBLE algorithm on $\mathcal{G}_j$ (i.e., the set F in Line 13 of Algorithm 1). Let $\mathcal{F} := \bigcup_{j=1}^{\eta} \mathcal{F}_j$ be the collection of these failed edges over all $j \in [\eta]$, and let $\mathcal{G}_\mathcal{F} := (V, \mathcal{F})$. We color the edges in $\mathcal{G}_\mathcal{F}$ using the folklore algorithm from Section 1.1, and an extra set of $O(\Delta(\mathcal{G}_\mathcal{F}))$ colors.

It is easy to verify that the above algorithm returns a proper edge coloring of G, and that the number of colors it uses is at most: $O(\Delta(\mathcal{G}_\mathcal{F})) + \sum_{j=1}^{\eta} |\mathcal{C}_j| = O(\Delta(\mathcal{G}_\mathcal{F})) + (1+\epsilon)^2\Delta' \cdot \eta = O(\Delta(\mathcal{G}_\mathcal{F})) + (1+\epsilon)^2\Delta' \cdot (\Delta/\Delta') = O(\Delta(\mathcal{G}_\mathcal{F})) + (1+\epsilon)^2\Delta$, w.h.p. We upper bound $\Delta(\mathcal{G}_\mathcal{F})$ in Claim 2.20 below. This implies the main result in this section, which is summarized in Corollary 2.21.

Claim 2.20. *We have $\Delta(\mathcal{G}_\mathcal{F}) = O(\epsilon\Delta)$ w.h.p.*

Proof. Throughout the proof, fix any node $v \in V$. Partition the set of indices $[\eta]$ into two subsets - $J_g := \{j \in [\eta] : v \in \mathcal{U}_j\}$ and $J_b := \{j \in [\eta] : v \in \mathcal{B}_j\}$ - depending on whether or not v is a good node in the corresponding subsampled graph. For each $j \in J_g$, Claim 2.18 and Lemma 2.17 imply that $\deg_{\mathcal{F}_j}(v) = O(\epsilon\Delta')$ w.h.p. Thus, summing over all $j \in J_g$, we derive the following bound w.h.p.

$$\sum_{j \in J_g} \deg_{\mathcal{F}_j}(v) = |J_g| \cdot O(\epsilon\Delta') \leq \eta \cdot O(\epsilon\Delta') = O(\epsilon\Delta). \tag{7}$$

Next, observe that every edge $(u,v) \in \bigcup_{j \in J_b} \mathcal{F}_j$, by definition, contributes to the set $E^\star$. Thus, from Claim 2.19, we infer that w.h.p.

$$\sum_{j \in J_b} \deg_{\mathcal{F}_j}(v) \leq \deg_{G^\star}(v) = o(\Delta). \tag{8}$$

From Equation (7) and Equation (8), we get $\deg_{\mathcal{G}_\mathcal{F}}(v) = O(\epsilon\Delta)$ w.h.p. The claim now follows from a union bound over all $v \in V$. □

Corollary 2.21. *W.h.p., the above algorithm returns a $(1+O(\epsilon))\Delta$-coloring of the input graph G.*

It is easy to verify that the subsampling step can be implemented in $O(m)$ time, because all we need to do is decide for each edge $e \in E$ which subgraph $\mathcal{G}_j$ it should appear in. We now implement the NIBBLE algorithm in each subsampled graph $\mathcal{G}_j$ using the approach outlined in Section 2.2.2. Putting everything together, this implies a $O_\epsilon(m)$ time algorithm for $(1+\epsilon)\Delta$-edge coloring in a general graph. See Appendix D for details.

2.4.1 Proof (Sketch) of Claim 2.19

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

Lemma 2.22. *Let G' be a subgraph of G obtained by sampling each edge in G independently with probability D/Δ, where $D \geq 2$. Then the probability that the g-neighborhood of a node u contains a cycle in G' is at most $3D^{5g}/\Delta$.*

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

$$\deg_{G^\star}(u) \leq \sum_{j \in [\eta]} X_j^u \cdot |N(u) \cap \mathcal{E}_j| \leq \sum_{j \in [\eta]} X_j^u \cdot \Delta(\mathcal{G}_j). \tag{9}$$

Each edge $e \in E$ is sampled in $\mathcal{G}_j$ independently with probability $1/\eta = \Delta'/\Delta$, and we have that $\Delta' \geq 2$ from the definitions of our parameters. Thus, setting $D = \Delta'$ and $g = T + 2$, Lemma 2.22 gives us: $\mathbb{E}\left[X_j^u\right] = \Pr\left[X_u^j = 1\right] \leq 3(\Delta')^{5(T+3)}/\Delta = o(1)$. By linearity of expectation, we get:

$$\mathbb{E}\left[\sum_{j \in [\eta]} X_j^u\right] = \eta \cdot o(1) = o(\Delta/\Delta'). \tag{10}$$

With some extra effort, we can show that the random variables $\{X_j^u\}_j$ are negatively associated (see Definition E.5). This allows us to apply a Chernoff bound w.r.t. Equation (10), and derive that: $\sum_{j \in [\eta]} X_j^u = o(\Delta/\Delta')$ w.h.p. Next, by Claim 2.18, w.h.p. we have: $\Delta(\mathcal{G}_j) = O(\Delta')$ for all $j \in [\eta]$. These last two observations, along with Equation (9), imply that $\deg_{G^\star}(u) = o(\Delta)$ w.h.p. The claim now follows from a union bound over all nodes $u \in V$.

3 Overview of our Dynamic Algorithm

In this section, we present an overview of our dynamic algorithm for edge coloring (see Theorem 1.2). As in Section 2, to convey the main insights behind our approach, the arguments here will be often informal and lacking in low-level details. The complete, formal proofs from this section are deferred to Appendix B and Appendix C.

To simplify the presentation, throughout Section 3 we will assume that the input graph $G = (V, E)$ always remains a forest with maximum degree at most Δ, throughout the sequence of updates. We will essentially dynamize the static algorithm from Section 2.1. Thus, it is reasonable to expect that once we have a fast dynamic implementation of Algorithm 1 on forests, we can extend this to work on general graphs using the subsampling framework from Section 2.4.[8]

[8]Note that it is very easy to implement the subsampling based partitioning from Section 2.4 in a dynamic setting: When an edge e is inserted, just sample an index $j \in [\eta]$ u.a.r. and add e to the subgraph $\mathcal{G}_j$.

Organization. In Section 3.1, we present our dynamic algorithm and analyze its *recourse* – the number of changes it makes to the colors of edges per update – ignoring any concern about efficient data structures (see Theorem 3.1). Section 3.2 shows that a slightly modified version of the dynamic algorithm from Section 3.1 can be implemented with fast data structures in $O_\epsilon(1)$ update time.

3.1 Bounding the Recourse

We start by presenting our dynamic algorithm.

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

Handling the sequence of updates. We use the superscript t to denote the status of any object at time t. For instance, $G^{(t)} = \left(V, E^{(t)}\right)$ denotes the input graph just after the t^{th} update. We maintain the tentative coloring $\tilde{\chi}^{(t)} : E^{(t)} \to \mathcal{C}$, which is obtained by executing Algorithm 1 on $G^{(t)}$, with the same random choices that were fixed at preprocessing. We also collect the subgraph $G_F^{(t)} := \left(V, F^{(t)}\right)$ consisting of all the failed edges and maintain a $O\left(\Delta(G_F^{(t)})\right)$-coloring $\psi^{(t)}$ in $G_F^{(t)}$ using an extra palette of colors that is mutually disjoint with $\mathcal{C}$. The final coloring χ is then defined as follows. For all $e \in E^{(t)}$, we have $\chi^{(t)}(e) = \tilde{\chi}^{(t)}(e)$ if $e \notin F^{(t)}$, and $\chi^{(t)}(e) = \psi^{(t)}(e)$ otherwise.

Theorem 3.1. *The dynamic algorithm presented above has an expected recourse of $O(1/\epsilon^4)$ per update, and at each time t w.h.p. maintains a proper $(1+O(\epsilon))\Delta$-edge coloring of $G^{(t)}$.*

We devote the rest of Section 3.1 to the proof of Theorem 3.1. Towards this end, first observe that the total number of colors used by the algorithm is $|\mathcal{C}| + O\left(\Delta(G_F^{(t)})\right) = (1+\epsilon)\Delta + O\left(\Delta(G_F^{(t)})\right)$, which equals $(1+O(\epsilon))\Delta$ w.h.p., according to Lemma 2.3. Thus, it now remains to bound the expected recourse of the algorithm. Let $A^{(t)} := \left\{e \in E^{(t-1)} \cup E^{(t)} : \tilde{\chi}^{(t-1)}(e) \neq \tilde{\chi}^{(t)}(e)\right\}$ denote the set of edges that change their tentative color due to the t^{th} update, where we define $\tilde{\chi}^{(t)}(e) = \perp$ for all $e \notin E^{(t)}$. Theorem 3.1 then follows from Lemma 3.2 and Lemma 3.3.

Lemma 3.2. *The recourse of the algorithm at time t is at most $O(1) + O\left(\left|A^{(t)}\right|\right)$.*

Proof. W.l.o.g., suppose that the t^{th} update involves the insertion of an edge $e^\star$ (we can analyze deletions analogously). Thus, we have $E^{(t)} := E^{(t-1)} \cup \{e^\star\}$. Note that $\chi^{(t')}(e) \in \{\tilde{\chi}^{(t')}(e), \psi^{(t')}(e)\}$ for all $e \in E^{(t)}$ and $t' \in \{t-1, t\}$.[9] Let $R^{(t)}$ be the recourse of the algorithm for the t^{th} update, and let $R_{\tilde{\chi}}^{(t)}$ and $R_\psi^{(t)}$ respectively denote the number of changes to the colorings $\tilde{\chi}$ and ψ due to the t^{th} update. Thus, we have $R^{(t)} \leq R_{\tilde{\chi}}^{(t)} + R_\psi^{(t)} = |A^{(t)}| + R_\psi^{(t)}$. It now remains to show that $R_\psi^{(t)} = O(1) + O\left(|A^{(t)}|\right)$. Towards this end, we first observe that $R_\psi^{(t)} \leq |F^{(t-1)} \oplus F^{(t)}|$,[10] because it

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

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

is trivial to maintain a $O(\Delta)$-coloring in a dynamic graph with maximum degree Δ with a recourse of at most one per update. The lemma now follows from Equation (11).

$$|F^{(t-1)} \oplus F^{(t)}| = O(1) + O\left(|A^{(t)}|\right). \tag{11}$$

We devote the rest of the proof towards showing why Equation (11) holds. Consider any subset $E' \subseteq \binom{V}{2}$ and any coloring $\chi' : E' \to \mathcal{C} \cup \{\perp\}$. We say that an $f \in E'$ is a *failed edge* w.r.t. (E', χ') iff either $\chi'(f) = \perp$ or it has a neighboring edge $e \in E'$ (i.e., e and f shares an endpoint) with the same color (i.e., $\chi'(e) = \chi'(f)$). For instance, $F^{(t)}$ is the set of failed edges w.r.t. $(E^{(t)}, \tilde{\chi}^{(t)})$.

Let $A^{(t)} \cup \{e^\star\} := \{e_1, \dots, e_\ell\}$, and for each $r \in [0, \ell]$, let $F(r)$ be the set of failed edges w.r.t. $E^{(t)}$ and the coloring where each edge $e \in \{e_1, \dots, e_r\}$ receives the color $\tilde{\chi}^{(t)}$ and each edge $e \in E^{(t)} \setminus \{e_1, \dots, e_r\}$ receives the color $\tilde{\chi}^{(t-1)}$. Intuitively, consider a process where we start with the coloring $\tilde{\chi}^{(t-1)}$, and then change the colors of the edges $\{e_1, \dots, e_\ell\}$ from $\tilde{\chi}^{(t-1)}$ to $\tilde{\chi}^{(t)}$, one at a time. The sets $F(r)$ track how $F^{(t-1)}$ evolves into $F^{(t)}$ during this process. It is easy to see that

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

since each $f \in F^{(t-1)} \oplus F^{(t)}$ must either be added to or removed from the set of failed edges after some edge $e \in \{e_1, \dots, e_\ell\}$ changes its color during the process described above. Now, fix any $r \in [\ell]$, let $e_r = (u, v)$, and consider the event when we change the color of e_r from $\tilde{\chi}^{(t-1)}(e_r)$ to $\tilde{\chi}^{(t)}(e_r)$ during the above process (assume for now that neither color is $\perp$). Due to this event, at most 2 edges can get added to the set F. This is because the only edges that will be added to F after this color change are edges incident to u and v with color $\tilde{\chi}^{(t)}(e_r)$ that are not already in F, and there can be at most one such edge incident on each of u and v. By an analogous argument, at most 2 edges can get removed from F due to this event. It follows that $|F(r-1) \oplus F(r)| \le 4$. A similar argument shows that if $\perp \in \left\{\tilde{\chi}^{(t-1)}(e_r), \tilde{\chi}^{(t)}(e_r)\right\}$, then $|F(r-1) \oplus F(r)| \le 3$. Thus, in both cases, we have $|F(r-1) \oplus F(r)| = O(1)$. Summing over all $r \in [\ell]$, this gives us $|F^{(t-1)} \oplus F^{(t)}| = O(\ell)$. Equation (11) now follows because $\ell \le 1 + |A^{(t)}|$. □

Lemma 3.3. *At each time t, we have $\mathbb{E}\left[|A^{(t)}|\right] = O(1/\epsilon^4)$.*

Proof. As in the proof of Lemma 3.2, w.l.o.g. suppose that the t^{th} update involves the insertion of an edge $e^\star$ (we can analyze deletions analogously). Thus, we have $E^{(t)} := E^{(t-1)} \cup \{e^\star\}$. Let $A_i^{(t)} := A^{(t)} \cap S_i^{(t)}$ for all $i \in [T]$. For the rest of the proof, fix the rounds $\{i_e\}_e$ of all potential edges $e \in \binom{V}{2}$, and condition on the high-probability event $\mathcal{Z}$ (see Lemma 2.5) on both $G^{(t-1)}$ and $G^{(t)}$.

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

$$\mathbb{E}\left[|A_i^{(t)}|\right] \le 4\epsilon \cdot \mathbb{E}\left[|A_{<i}^{(t)}|\right] \text{ for all } i \in [i_{e^\star} + 1, T]. \tag{12}$$

Equation (12) implies that $\mathbb{E}[|A^{(t)}|] \le (1 + 4\epsilon)^T \le e^{4\epsilon T} = O(1/\epsilon^4)$. Thus, it now remains to prove Equation (12). Towards this end, consider the following scenario where: (i) we fixed the rounds $\{i_e\}_e$ and color-sequences $\{c_e\}_e$ of all potential edges at preprocessing, (ii) we have already computed the outcome of Algorithm 1 on $G^{(t-1)}$ in accordance with these choices $\{i_e, c_e\}_e$, and (iii) now we are running Algorithm 1 again on $G^{(t)}$ with the same choices $\{i_e, c_e\}_e$ and observing which edges get added to $A^{(t)}$ as Algorithm 1 implements the rounds $i = 1, \dots, T$ on $G^{(t)}$, in this order.

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

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

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

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

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

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

$$|A_i^{(t)}| \ \leq \sum_{e' \in A_{<i}^{(t)}} \left(|\Gamma_i(e')| + |\Lambda_i(e')| \right) \text{ for all } i \in [i_{e^\star} + 1, T]. \tag{13}$$

We now make the following claim, whose proof is deferred to Section 3.1.1.

Claim 3.4. *Consider any round $i \in [i_{e^\star} + 1, T]$ and any edge $e \in S_{<i}^{(t)}$. Then we have:*

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

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

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

In the above derivation, the first inequality follows from Equation (13), whereas the second inequality follows from Claim 3.4. Thus, Equation (12) holds, and this concludes the proof of the lemma. □

3.1.1 Proof of Claim 3.4

The occurrence of the event $\left\{e \in A_{<i}^{(t)}\right\}$ depends only on the color-sequences of those edges $e' \in S_{<i}^{(t)}$ that are in the same connected component as e in $G_{<i}^{(t)} := (V, S_{<i}^{(t)})$. For the rest of the proof, we fix these relevant color-sequences in such a way that the event $\left\{e \in A_{<i}^{(t)}\right\}$ occurs. Below, we will show that $\mathbb{E}[|\Gamma_i(e)|] \leq 2\epsilon$. The proof for $\mathbb{E}[|\Lambda_i(e)|] \leq 2\epsilon$ is completely analogous, and taken together, they imply Claim 3.4.

By linearity of expectation, we have:

$$\mathbb{E}\left[|\Gamma_i^{(t)}(e)|\right] = \sum_{f \in N_i^{(t)}(e)} \Pr\left[\tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e)\right]. \tag{14}$$

Let $e = (u, v)$. The palettes $P_i^{(t-1)}(u)$ and $P_i^{(t-1)}(v)$ are completely determined by the color-squences we have fixed, because $G_{<i}^{(t-1)}$ is a subgraph of $G_{<i}^{(t)}$ as we are considering the scenario where the t^{th} update is an edge-insertion. Consider an edge $f = (u, w) \in N_i^{(t-1)}(e)$ (note that $N_i^{(t)}(e) = N_i^{(t-1)}(e)$ since $i_{e^\star} < i$). Since the graph $G_i^{(t)}$ contains no cycles, e and w lie in different connected components of $G_{<i}^{(t)}$. Thus, the color $\tilde{\chi}^{(t)}(e)$ is independent of the palette $P_i^{(t-1)}(w)$. Let $c = \tilde{\chi}^{(t)}(e)$. We can now apply Lemma 2.2 and Corollary 2.6 to get:

$$\Pr\left[\tilde{\chi}^{(t-1)}(f) = c\right] \leq \frac{1}{|P_i^{(t-1)}(u)|} \leq \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}. \tag{15}$$

From Equation (14) and Equation (15), we now derive that:

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

The penultimate step in the above derivation follows from Lemma 2.5.

3.2 Bounding the Update Time

We now outline how to implement a modified version of the algorithm in Section 3.1, which leads to dynamic $(1+\epsilon)\Delta$-edge coloring in $O(\text{poly}(1/\epsilon))$ update time (see Theorem 1.2).

A "template" algorithm. We start with a *template algorithm*, which differs from the one in Section 3.1 in only one aspect: The color-sequence of every potential edge is now of length $K := \lceil (8/\epsilon^2) \log(1/\epsilon) \rceil$. Specifically, at preprocessing each potential edge e constructs the color-sequence c_e by sampling K, as opposed to infinitely many, colors $c_e(1), \ldots, c_e(K)$ from the palette $\mathcal{C} = [(1+\epsilon)\Delta]$, independently and u.a.r. Subsequently, the algorithm handles the graph $G^{(t)}$ at time t as follows. While executing round $i \in [T]$ of the NIBBLE algorithm on $G^{(t)}$, an edge $e \in S_i^{(t)}$ picks the minimum index $\ell \in [K]$ s.t. $c_e(\ell) \in P_i^{(t)}(u) \cap P_i^{(t)}(v)$, and sets $\tilde{\chi}^{(t)}(e) := c_e(\ell)$ and $\ell_e^{(t)} := \ell$. If there is no such ℓ, then it sets $\tilde{\chi}^{(t)}(e) := \perp$ and $\ell_e^{(t)} := 0$. Everything else remains the same as in Section 3.1. We now give an intuitive justification as to why the above modification does not (meaningfully) change the guarantees derived in Section 3.1 (see Appendix A for a formal argument).

An intuitive justification. Fix the rounds $\{i_e\}_e$ of all potential edges $e \in \binom{V}{2}$, and condition on the high-probability event $\mathcal{Z}$ (see Lemma 2.5) on the current graph $G^{(t)}$. Consider any round $i \in [T]$ and any edge $e \in S_i^{(t)}$. By Corollary 2.8, we have that $|P_i(e)| \geq \epsilon^2(1+\epsilon)\Delta/8$ w.h.p. As $|\mathcal{C}| = (1+\epsilon)\Delta$, the probability that the color-sequence c_e does not contain some color from $P_i(e)$ is given by: $\left(1 - \frac{\epsilon^2(1+\epsilon)\Delta}{8 \cdot |\mathcal{C}|}\right)^K \leq \left(1 - \frac{\epsilon^2}{8}\right)^K \leq \left(1 - \frac{\epsilon^2}{8}\right)^{(8/\epsilon^2)\log(1/\epsilon)} \leq \epsilon$. To summarize, with probability at most ϵ, the template algorithm mistakenly sets $\tilde{\chi}^{(t)}(e) = \perp$. But with probability $1-\epsilon$, it correctly sets $\tilde{\chi}^{(t)}(e)$ to be a color chosen u.a.r. from $P_i^{(t)}(e)$. Thus, the template algorithm is almost similar to the original algorithm, except that each edge $e \in E^{(t)}$, before even being considered for receiving a tentative color, gets added to the failed set $F^{(t)}$ with probability ϵ. This increases the maximum degree of the subgraph $G_F^{(t)} := (V, F^{(t)})$ by at most $\epsilon\Delta$ (we can show that this bound holds w.h.p.). Since anyway we maintain a $O\left(G_F^{(t)}\right)$-coloring in $G_F^{(t)}$ using a separate palette, the total number of colors needed by the algorithm continues to remain $(1 + O(\epsilon))\Delta$.

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

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

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

Claim 3.5. *For each $v \in V, i \in [T]$ and $c \in \mathcal{C}$, we have $\mathbb{E}\left[|\mathcal{L}_{v,i}^{(t)}(c)|\right] = K = O_\epsilon(1)$.*

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

Claim 3.6. *We always have $\sum_{v \in V, i \in [T], c \in \mathcal{C}} |\mathcal{L}_{v,i}^{(t)}(c)| = O\left(KT \cdot |E^{(t)}|\right) = O_\epsilon\left(|E^{(t)}|\right)$. Further, insertion/deletion of an edge e in G can lead to at most $2KT = O_\epsilon(1)$ changes in the sets $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$.*

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

Claim 3.6 implies that we can maintain the hash-tables for $\left\{\mathcal{L}_{v,i}^{(t)}(c)\right\}_{v,i,c}$ in $O_\epsilon(1)$ update time (note that these hash-table don't depend at all on the tentative coloring $\tilde{\chi}^{(t)}$), and that the total space complexity of this data structure is $O_\epsilon\left(|E^{(t)}|\right)$. We now show how using this data structure we can implement the template algorithm in expected update time proportional to its recourse, which in turn, is $O(1/\epsilon^4)$ according to Theorem 3.1.

Modifying the coloring $\tilde{\chi}$ after the t^{th} update. Recall the notations from Section 3.1, and consider the following scenario. We already have the coloring $\tilde{\chi}^{(t-1)}$ on $G^{(t-1)}$, when we receive the

t^{th} update to G. Our goal now is to change the coloring $\tilde{\chi}^{(t-1)}$ into $\tilde{\chi}^{(t)}$.[11] To achieve this goal, all we need to do is consider the rounds $i = 1, \ldots, T$, one at a time. And while we are at round $i \in [T]$, we need to identify the set $A_i^{(t)}$ and change the color of each edge $e \in A_i^{(t)}$ appropriately.

We claim that once we have identified an edge $e \in A_i^{(t)}$ during round i, changing its color takes only $O(K) = O_\epsilon(1)$ time. To see why this is true, note that we can easily maintain the complements of the palettes $\left\{\overline{P_i(v)}\right\}_v$ for all nodes $v \in V$ as hash-tables, where $\overline{P_i(v)} := \mathcal{C} \setminus P_i(v)$.[12] Now, for the edge $e = (u, v) \in A_i^{(t)}$, we scan through its color-sequence c_e and identify the smallest index (if any) $\ell \in [K]$ such that $c_e(\ell) \notin \overline{P_i^{(t)}(u)} \cup \overline{P_i^{(t)}(v)}$. This takes $O_\epsilon(1)$ time per index in $[K]$, because at the start of round i we already have access to the hash tables for $\overline{P_i^{(t)}(u)}$ and $\overline{P_i^{(t)}(v)}$.

Thus, the time spent on implementing round i is dominated by the time spent on identifying the set $A_i^{(t)}$ in the first place. We will now show that using our data structures we can perform this task in expected time $O_\epsilon\left(|A_{<i}^{(t)}|\right)$. Summing over $i \in [T]$, this will lead to an expected update time of $\sum_{i \in [T]} O_\epsilon\left(|A_{<i}^{(t)}|\right) = O_\epsilon\left(\sum_{i \in [T]} T \cdot |A_i^{(t)}|\right) = O_\epsilon(|A^{(t)}|)$, which is $O_\epsilon(1)$ in expectation, according to Lemma 3.3. It now only remains to prove the claim below.

Claim 3.7. *While implementing round i of the template algorithm after the t^{th} update, we can identify the set $A_i^{(t)}$ in expected time $O_\epsilon(|A_{<i}^{(t)}|)$.*

Proof. (Sketch) From the proof of Lemma 3.3 (specifically, from the discussion in the paragraph preceding Equation (13)), it follows that an edge $e \in S_i^{(t)}$ belongs to $A_i^{(t)}$ only if it has a neighbor $e' \in N_{<i}^{(t)}(e) \cap A_{<i}^{(t)}$ such that either $e \in \Gamma_i^{(t)}(e')$ or $e \in \Lambda_i^{(t)}(e')$. Note that in the former (resp. latter) case, the color $\tilde{\chi}^{(t)}(e')$ (resp. $\tilde{\chi}^{(t-1)}(e')$) must appear at least once in the color-sequence c_e. In other words, an edge $e = (u, v) \in S_i^{(t)}$ appears in $A_i^{(t)}$ only if it has a neighboring edge $e' = (v, w) \in N_{<i}^{(t)}(e) \cap A_{<i}^{(t)}$ such that $e \in \mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t)}(e')\right) \cup \mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t-1)}(e')\right)$. This motivates the following approach for identifying the set $A_i^{(t)}$ at the start of round i. Construct a set $\mathcal{A}_i^{(t)}$ by taking the union, over all $e' = (v, w) \in A_{<i}^{(t)}$, of the sets $\mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t)}(e')\right) \cup \mathcal{L}_{v,i}^{(t)}\left(\tilde{\chi}^{(t-1)}(e')\right)$. By Claim 3.5, this takes $O_\epsilon\left(|A_{<i}^{(t)}|\right)$ expected time. We also know for sure that $A_i^{(t)} \subseteq \mathcal{A}_i^{(t)}$, and $\mathbb{E}\left[|\mathcal{A}_i^{(t)}|\right] = O_\epsilon\left(|A_{<i}^{(t)}|\right)$.

We now scan through all the edges $e \in \mathcal{A}_i^{(t)}$, and for each of them determine whether or not it belongs to $A_i^{(t)}$. This takes $O(K) = O_\epsilon(1)$ expected time per edge in $\mathcal{A}_i^{(t)}$, because given an edge $e = (u, v) \in \mathcal{A}_i^{(t)}$ all we need to do is to check which of the colors $\{c_e(\ell)\}_{\ell \in [K]}$ belong to $P_i^{(t)}(u) \cap P_i^{(t)}(v) = \mathcal{C} \setminus \left(\overline{P_i^{(t)}(u)} \cup \overline{P_i^{(t)}(v)}\right)$, and this takes $O(1)$ expected time per color.

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

Getting rid of the preprocessing time. Suppose that we start with an empty graph $G^{(0)} := (V, E^{(0)})$ before any update arrives. We can get rid of the $O_\epsilon(n^2)$ preprocessing by making the following simple observation: We don't need to fix the rounds and color-sequences of potential edges in advance. We can do that *on the fly*. Specifically, when an edge e gets inserted, we sample its round i_e and color-sequence c_e in $O_\epsilon(1)$ time. We work with (i_e, c_e) as long as the edge e is

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

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

present in the input graph. When the edge e gets deleted, we remove all information/storage about (i_e, c_e). If the edge e gets inserted again at some point in future, then we sample a fresh pair (i_e, c_e) and work with this new sample. It is easy to verify that all our recourse and update time bounds, as well as the guarantee on the total number of colors used, continue to hold if we make this simple modification. The only effect it has is that now we don't need to worry about preprocessing time at all, provided we are starting with an empty graph.

References

[AS21] Sepehr Assadi and Shay Solomon. Fully dynamic set cover via hypergraph maximal matching: An optimal approximation through a local approach. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 8:1–8:18, 2021. ↑1

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *Integer Programming and Combinatorial Optimization (IPCO)*, volume 10328 of *Lecture Notes in Computer Science*, pages 86–98, 2017. ↑1

[BCHN18] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018. ↑1

[BGK⁺22] Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic $(\Delta + 1)$-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022. ↑1

[BGM17] Sayan Bhattacharya, Manoj Gupta, and Divyarthi Mohan. Improved algorithm for dynamic b-matching. In *25th Annual European Symposium on Algorithms (ESA)*, volume 87 of *LIPIcs*, pages 15:1–15:13, 2017. ↑1

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2830–2842. SIAM, 2021. ↑2, ↑4, ↑5, ↑56

[BHNW21] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2537–2549, 2021. ↑1

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1872–1885. SIAM, 2019. ↑1

[BM17] 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. ↑1

[Bro11] Daniel G. Brown. How i wasted too long finding a concentration inequality for sums of geometric variables. *Unpublished Manuscript, University of Waterloo*, 2011. ↑55

[CHL+18] 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. ↑2

[Chr23] Aleksander Bjørn Grodt Christiansen. The power of multi-step vizing chains. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1013–1026, 2023. ↑1, ↑2

[DadH90] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Automata, Languages and Programming: 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings 17*, pages 6–19. Springer, 1990. ↑55

[DGP98] Devdatt P. Dubhashi, David A. Grable, and Alessandro Panconesi. Near-optimal, distributed edge colouring via the nibble method. *Theor. Comput. Sci.*, 203(2):225–251, 1998. ↑2

[DHZ19] 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. ↑1, ↑2, ↑5

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, 2009. ↑56, ↑57

[DR96] Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996. ↑57

[GNK+85] Harold N. Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. Algorithms for edge-coloring graphs. *Technical report, Technical report 41/85, Tel Aviv University*, 1985. ↑2

[HP20] Monika Henzinger and Pan Peng. Constant-time dynamic $(\Delta+1)$-coloring. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 53:1–53:18, 2020. ↑1

[JDP83] Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, pages 286–295, 1983. ↑57

[KLS+22] Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 104–116. ACM, 2022. ↑3, ↑4, ↑5, ↑11, ↑13, ↑32

[Lar12] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 85–94, 2012. ↑1

[PD06] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. ↑1

[PS16] David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016. ↑1

[Sin19] Corwin Sinnamon. A randomized algorithm for edge-colouring graphs in $o(m\sqrt{n})$ time. *CoRR*, abs/1907.03201, 2019. ↑2

[Sol16] Shay Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016. ↑1

[SW18] Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *LIPIcs*, pages 72:1–72:16, 2018. ↑1

[Viz64] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964. ↑1

Acknowledgements

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).

A Our Static Algorithm (Full Version)

Throughout this paper, we fix some constant ϵ such that $0 < \epsilon \leq 1/10$, and define parameters $T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ and $K := \lceil (8/\epsilon^2)\log(1/\epsilon) \rceil$. In this appendix, we describe our static algorithm in full detail and provide a complete analysis. The main result in this appendix is Theorem A.2, which is restated below.

Theorem A.1. *Let $\epsilon \in (0, 1/10)$ be a constant. Then, given a graph G and a parameter $\Delta \geq (100\log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ, the algorithm* StaticColor *produces a $(1+61\epsilon)\Delta$-edge coloring of G with probability at least $1 - 8/n^6$.*

A.1 Algorithm Description

The algorithm StaticColor**.** Our main algorithm, StaticColor, takes as input a graph $G = (V, E)$ with n nodes, and a parameter $\Delta \in \mathbb{N}$. The algorithm uses two subroutines: Partition and Nibble. We first use the algorithm Partition to split the input graph G into $\eta = \lceil \Delta^{1-1/(30T)} \rceil$ many graphs $\mathcal{G}_1, ..., \mathcal{G}_\eta$ by assigning the edges of G to one of the $\mathcal{G}_j$ independently and uniformly at random. We then proceed to call the algorithm Nibble on each of the $\mathcal{G}_j$, where we use $(1+\epsilon)^2\Delta^{1/(30T)}$ many colors to color most of the edges in $\mathcal{G}_j$. Finally, we take all of the edges that failed to be colored by our calls to Nibble and greedily color them. Algorithm 2 gives the pseudocode for StaticColor.

Algorithm 2 StaticColor(G, ϵ)

1: $\mathcal{G}_1, \ldots, \mathcal{G}_\eta \leftarrow$ Partition$(G, \Delta, \Delta^{1/(30T)})$
2: **for** $j = 1, ..., \eta$ **do**
3: $\quad \mathcal{F}_j \leftarrow$ Nibble$(\mathcal{G}_j, (1+\epsilon)\Delta^{1/(30T)}, \epsilon)$
4: Greedily color the edges in $H \leftarrow \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_\eta$ using $3\Delta(H)$ many colors

For analytic purposes, and also to make the algorithm easier to present and dynamize, it will be useful for us to fix the randomness used by our algorithm while making calls to Partition and Nibble in advance. Hence, we will describe how all the relevant randomness is generated in advance while describing each part of our algorithm. The assumption can be removed later.

The algorithm Partition**.** This algorithm takes as input a graph G and some parameters Δ and Δ' such that $\Delta' \leq \Delta$, and outputs a partition $\mathcal{G}_1, ..., \mathcal{G}_\eta$ of G, where $\eta = \lceil \Delta/\Delta' \rceil$, obtained by assigning the edges of G to one of the $\mathcal{G}_j$ independently and uniformly at random. The node sets $V(\mathcal{G}_j)$ in all of the $\mathcal{G}_j$ are the same as the node set V of the graph G. In order to fix the randomness in advance, for each *potential edge* of the graph $e \in \binom{V}{2}$, we sample an index $j_e \in [\eta]$ independently and uniformly at random, and place e into the graph $\mathcal{G}_{j_e}$. Notice that after fixing the random variables $\{j_e\}_e$ the partition of the graph depends only on what edges are present in G. Algorithm 3 gives the pseudocode for Partition.

Algorithm 3 Partition(G, Δ, Δ')

1: $\eta \leftarrow \lceil \Delta/\Delta' \rceil$
2: $\mathcal{E}_j \leftarrow \varnothing$ for all $j \in [\eta]$
3: **for** $e \in E$ **do**
4: $\quad$ Place e into one of $\mathcal{E}_1, \ldots, \mathcal{E}_\eta$ independently and u.a.r.
$\mathcal{G}_j \leftarrow (V, \mathcal{E}_j)$ for all $j \in [\eta]$
5: **return** $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$

The algorithm Nibble**.** This algorithm takes as input a graph G, a parameter Δ, and uses a palette $\mathcal{C}$ of $\lceil (1+\epsilon)\Delta \rceil$ colors. The Nibble algorithm runs for $T := \lfloor (1/\epsilon)\log(1/\epsilon) \rfloor$ *rounds*.

At the start of round $i \in [T]$, we have a subset of edges $E_i \subseteq E$ such that the algorithm has already assigned tentative colors to the remaining edges $E \setminus E_i$. We denote this *tentative* partial coloring by $\tilde{\chi} : E \setminus E_i \to \mathcal{C} \cup \{\perp\}$, which need not necessarily be proper. For each node $u \in V$, we refer to the set of colors $P_i(u) := \mathcal{C} \setminus \tilde{\chi}(N(u) \setminus E_i)$ as the *palette* of u at the start of round i, where $N(u) \subseteq E$ denotes the set of edges incident on u in G. In words, the palette $P_i(u)$ consists of the set of colors that were *not* tentatively assigned to any incident edge of u in previous rounds. We also define $P_i(u, v) := P_i(u) \cap P_i(v)$ to be the *palette* of any edge $(u, v) \in E_i$ at the start of round i.

We start by initializing $E_1 \leftarrow E$, and $P_1(u) \leftarrow \mathcal{C}$ for all $u \in V$. Subsequently, for $i = 1, \ldots, T$, we implement round i as follows. Each edge $e \in E_i$ *selects* itself independently with probability ϵ. Let $S_i \subseteq E_i$ be the set of selected edges. Next, in parallel, each edge $e \in S_i$ samples $K := \lceil (8/\epsilon^2)\log(1/\epsilon) \rceil$ colors $c_1, \ldots, c_K$ independently and uniformly at random from $\mathcal{C}$ and sets its tentative color $\tilde{\chi}(e)$ to some c_ℓ which is contained in $P_i(e)$ if such a color exists. Otherwise, we set $\tilde{\chi}(e) \leftarrow \perp$. At this point, we define the collection $F_i \subseteq S_i$ of *failed* edges in round i. We say that an edge $e = (u, v) \in S_i$ *fails* in round i iff either (i) $\tilde{\chi}(e) = \perp$, or (ii) there is a neighboring edge

$f \in (N(u) \cup N(v)) \cap S_i$ which was also selected in round i and received the same tentative color as the edge e (i.e., $\tilde{\chi}(e) = \tilde{\chi}(f)$). Let $F_i \subseteq S_i$ denote this collection of failed edges (in round i). We now set $E_{i+1} \leftarrow E_i \setminus S_i$ and proceed to the next round $i+1$.

In order to fix the randomness in advance, for each potential edge of the graph e, we sample a round for the edge e independently from $\textsc{CappedGeo}(\epsilon, T+1)$, which we denote by i_e.[13] We also assume that for each potential edge e we have some colors $c_e(1), ..., c_e(K)$ where each $c_e(\ell)$ is sampled uniformly at random and independently from $\mathcal{C}$. We define ℓ_e to be $\min\{\ell \mid c_e(\ell) \in P_{i_e}(e)\}$ (taking the convention that $\min \varnothing = 0$) and note that $\tilde{\chi}(e) = c_e(\ell_e)$ (as long as $c_e \cap P_{i_e}(e) \neq \varnothing$). Algorithm 4 gives the pseudocode for $\textsc{Nibble}$.

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

Algorithm 4 $\textsc{Nibble}(G, \Delta, \epsilon)$

- $\chi(e) \leftarrow \perp$ and $\tilde{\chi}(e) \leftarrow \perp$ for all $e \in E(G)$
- $E_1 \leftarrow E(G)$
- **for** $i = 1, ..., T$ **do**
  - **for** $e \in S_i$ **do**
    - $\ell_e \leftarrow \min\{\ell \mid c_e(\ell) \in P_i(e)\}$
    - **if** $1 \leq \ell_e \leq K$ **then**
      - $\tilde{\chi}(e) \leftarrow c_e(\ell_e)$
  - $F_i \leftarrow \{e \in S_i \mid \exists f \in N(e) \cap S_i \text{ such that } \tilde{\chi}(f) = \tilde{\chi}(e)\} \cup \{e \in S_i \mid \tilde{\chi}(e) = \perp\}$
  - $\chi(e) \leftarrow \tilde{\chi}(e)$ for all $e \in S_i \setminus F_i$
  - $E_{i+1} \leftarrow E_i \setminus S_i$
- $F_{T+1} \leftarrow E_{T+1}$
- $F \leftarrow \bigcup_{i=1}^{T+1} F_i$
- **return** F

We now prove the following result which describes the behavior of $\textsc{StaticColor}$.

Theorem A.2. *Let $\epsilon \in (0, 1/10)$ be a constant. Then, given a graph G and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ, the algorithm $\textsc{StaticColor}$ produces a $(1+61\epsilon)\Delta$-edge coloring of G with probability at least $1 - 8/n^6$.*

A.2 Analysis on Locally Treelike Graphs

For the rest of Appendix A.2, fix some graph $G = (V, E)$ of maximum degree at most Δ such that $\Delta \geq (100 \log n)/\epsilon^4$. Let $\mathcal{N}_G(u, j) := G[\{v \in V : \text{dist}_G(u, v) \leq j\}]$ denote the j-hop neighborhood of any node $u \in V$.[14] We refer to $\mathcal{N}_G(u, j)$ as the j-neighborhood of u. Let $U \subseteq V$ be the set of nodes $u \in V$ such that the $(T+1)$-neighborhood of u is a tree. We refer to the nodes in U as *good* and the nodes in $V \setminus U$ as *bad*. Now suppose we run Algorithm 4 on input (G, Δ, ϵ). We now proceed to analyze the behavior of the algorithm on this input.

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

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

A.2.1 Locality of the Nibble Method

Lemma A.3. *Let $u \in V$, $i \in [T]$. Then $P_i(u)$ depends only on $\mathcal{N}(u, i-1)$.*

Proof. We first recall the way our algorithm generates random bits in advance. We assign each *potential edge* $e \in \binom{V}{2}$ a round i_e sampled from $\textsc{CappedGeo}(\epsilon, T+1)$ independently. Furthermore, we assign each potential edge a sequence of colors $c_e(1), \ldots c_e(K)$ generated by sampling colors independently and u.a.r from $\mathcal{C}$. We now argue inductively that the set $P_i(u)$ depends only on $\mathcal{N}(u, i-1)$, i.e. that the structure of the graph outside of the $(i-1)$-neighborhood of u has no effect on the palette of u during the first $i-1$ rounds of the nibble method.

Clearly, $P_1(u) = \mathcal{C}$ for all nodes u and hence does not depend on anything. Now, for the induction step, suppose that $P_i(u)$ depends only on $\mathcal{N}(u, i-1)$ for all nodes u. Given some node u, note that $P_{i+1}(u)$ depends on (i) the rounds of the edges incident on u, (ii) the sequences of colors assigned to the edges incident on u, and (iii) the palettes $P_i(v)$ of the nodes adjacent to u. Since (i) and (ii) are fixed in advance, $P_{i+1}(u)$ is completely determined by which nodes are its neighbors and their palettes at the end of iteration i. Hence, by the induction hypothesis, it follows that $P_{i+1}(u)$ depends on $\bigcup_{v \in N(u)} \mathcal{N}(v, i-1)$ which is exactly the i-neighborhood of u, $\mathcal{N}(u, i)$. $\square$

It immediately follows from Lemma A.3 that, for an edge e, the palette $P_i(e)$ depends only on $\mathcal{N}(e, i-1)$. Since the tentative color assigned to e depends only on the round i_e and the palette $P_{i_e}(e)$, it also follows that $\tilde{\chi}(e)$ depends only on $\mathcal{N}(e, i_e - 1) \subseteq \mathcal{N}(e, T-1)$. Hence, given some node u, the tentative colors assigned to edges in $N(u)$ depend only on $\mathcal{N}(N(u), T-1) \subseteq \mathcal{N}(u, T)$. Since the edges in $N(u)$ that fail depend on the tentative colors of the edges in $N^2(U) := N(N(U))$, while analyzing the palettes, tentative colors, and failure status of the edges in $N(u)$ we can assume that the input graph is $\mathcal{N}(N^2(u), T-1) = \mathcal{N}(u, T+1)$. In the case where u is good, this will allow us to significantly simplify the analysis.

A.2.2 The Symmetry of Algorithm 1

We now prove some lemmas that will be crucial for analyzing the types of colorings generated by our algorithm. For the rest of Appendix A.2.2 we fix the random bits that determine the rounds of all edges in G. The following lemma formalizes the main "symmetry" property of $\textsc{Nibble}$ which says that it treats all colors equally.

Lemma A.4. *For all $u \in V$, $i \in [T]$, $C \subseteq \mathcal{C}$, and permutations $\pi : \mathcal{C} \longrightarrow \mathcal{C}$, we have that*

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

Proof. We prove this lemma via a coupling argument. Let $\mathcal{A}$ denote the algorithm $\textsc{Nibble}$. Given some permutation π of $\mathcal{C}$, we define a new algorithm $\mathcal{A}^\pi$, which behaves in the exact same way as $\mathcal{A}$, except that given the color sequence c_e for the potential edge e, it uses the color sequences $\pi \circ c_e$ instead. We denote the palette $P_i(u)$ produced by algorithm $\mathcal{A}'$ by $P_i^{(\mathcal{A}')}(u)$. We now prove by induction that for all $u \in V$, $i \in [T]$, and permutations π of $\mathcal{C}$, we have that

$$P_i^{(\mathcal{A}^\pi)}(u) = \pi\left(P_i^{(\mathcal{A})}(u)\right). \tag{16}$$

Fix some permutation π of $\mathcal{C}$. It's easy to see that (16) holds for all $u \in V$ when $i = 1$ since the palettes all equal $\mathcal{C}$ and π is a permutation. Now suppose that (16) holds for some $i \in [T-1]$ and all $u \in V$. Then, for all $e = (u, v) \in S_i$, we have that

$$P_i^{(\mathcal{A}^\pi)}(e) = P_i^{(\mathcal{A}^\pi)}(u) \cap P_i^{(\mathcal{A}^\pi)}(v) = \pi\left(P_i^{(\mathcal{A})}(u)\right) \cap \pi\left(P_i^{(\mathcal{A})}(v)\right)$$

$$= \pi\left(P_i^{(\mathcal{A})}(u) \cap P_i^{(\mathcal{A})}(v)\right) = \pi\left(P_i^{(\mathcal{A})}(e)\right),$$

thus, it follows that

$$c_e(\ell) \in P_i^{(\mathcal{A})}(e) \text{ iff } \pi(c_e(\ell)) \in \pi\left(P_i^{(\mathcal{A})}(e)\right) \text{ iff } \pi(c_e(\ell)) \in P_i^{(\mathcal{A}^\pi)}(e).$$

We get that $\ell_e^{(\mathcal{A})} = \ell_e^{(\mathcal{A}^\pi)}$ for all $e \in S_i$, which implies that

$$\tilde{\chi}^{(\mathcal{A}^\pi)}(e) = \pi\left(c_e(\ell_e^{(\mathcal{A}^\pi)})\right) = \pi\left(c_e(\ell_e^{(\mathcal{A})})\right) = \pi\left(\tilde{\chi}^{(\mathcal{A})}(e)\right)$$

for all $e \in S_i$.[15] Finally, it follows that

$$\pi\left(P_{i+1}^{(\mathcal{A})}(u)\right) = \pi\left(P_i^{(\mathcal{A})}(u) \setminus \tilde{\chi}^{(\mathcal{A})}(N_i(u))\right) = \pi\left(P_i^{(\mathcal{A})}(u)\right) \setminus \pi\left(\tilde{\chi}^{(\mathcal{A})}(N_i(u))\right)$$
$$= P_i^{(\mathcal{A}^\pi)}(u) \setminus \tilde{\chi}^{(\mathcal{A}^\pi)}(N_i(u)) = P_{i+1}^{(\mathcal{A}^\pi)}(u).$$

This concludes the induction. By now noticing that algorithms $\mathcal{A}$ and $\mathcal{A}^\pi$ are actually the same (since applying a permutation to an independent uniform sample returns an independent uniform sample) we get that, for any $C \subseteq \mathcal{C}$,

$$\Pr\left[P_i^{(\mathcal{A})}(u) = C\right] = \Pr\left[P_i^{(\mathcal{A}^\pi)}(u) = C\right] = \Pr\left[\pi\left(P_i^{(\mathcal{A})}(u)\right) = C\right]$$

and the lemma follows. □

Fix any $i \in [T]$ and let X_c^w be the indicator for the event that $c \in P_i(w)$ for a color c and node w.

Lemma A.5. *Let $u \in V$. Given that $\sum_c X_c^u = \gamma$ for some $\gamma \in \mathbb{N}$, we have that $\{X_c^u\}_c$ is a permutation distribution (see Definition E.7) where exactly γ of the X_c^u are 1 and the rest are 0.*

Proof. Since the random variables X_c^u are all indicators and we know that $\sum_c X_c^u = \gamma$, it follows that exactly γ of them are 1 and the rest are 0. It then follows from Lemma A.4 that, for any $C, C' \subseteq \mathcal{C}$ of size γ, $\Pr[P_i(u) = C] = \Pr[P_i(u) = C']$, and hence this collection of random variables forms a permutation distribution. □

Lemma A.6. *Let $\mathfrak{X}_1, \ldots, \mathfrak{X}_\ell$ be families of random variables that depend on disjoint collections of random bits. Then these families of random variables are mutually independent.*

Lemma A.7. *Let $u_1, \ldots, u_\ell \in V$ be nodes that are mutually disconnected in the graph $(V, S_{<i})$. Then we have that the families of random variables $\{X_c^{u_1}\}_c, \ldots, \{X_c^{u_\ell}\}_c$ depend on disjoint collections of random bits.*

Lemma A.8. *Let $e = (u, v) \in S_i$, such that $e \in N^2(U)$ and u and v are not connected in the graph $(V, S_{<i})$. Suppose we fix the random bits used in the first $i-1$ rounds that determine the palette $P_i(u)$ (but not those that determine $P_i(v)$) and let c be a color that does not depend on the random bits that determine $P_i(u)$. Then we have that $\Pr[\tilde{\chi}(e) = c] \le 1/|P_i(u)|$.*

[15] Here $\ell_e^{(\mathcal{A}')}$ and $\tilde{\chi}^{(\mathcal{A}')}(e)$ denote the color index ℓ_e and tentative color $\tilde{\chi}(e)$ produced by algorithm $\mathcal{A}'$.

Proof. First, note that by Lemma A.3 we can assume that the input graph is $\mathcal{N}(e, T-1)$. Since $e \in N^2(U)$, $\mathcal{N}(e, T-1)$ is a subgraph of $\mathcal{N}(w, T+1)$ for some $w \in U$, and hence $\mathcal{N}(e, T-1)$ is a tree. It follows that u and v are not connected in the graph $(V, S_{<i})$. Note that since $P_i(u)$ and $P_i(v)$ are functions of $\{X_c^u\}_c$ and $\{X_c^v\}_c$ respectively, which depend on disjoint sets of random bits by Lemma A.7, it is possible to fix the random bits this way. In the event that e does not fail, $\tilde{\chi}(e)$ is a uniform sample from $P_i(u) \cap P_i(v)$ by the construction of our algorithm. By Lemma A.6, the families of random variables $\{X_c^u\}_c$ and $\{X_c^v\}_c$ are independent, so by Lemma A.5 we get that $P_i(v)$ is a uniform random subset of $\mathcal{C}$. Hence, as long as $P_i(u) \cap P_i(v) \neq \varnothing$, sampling a color uniformly at random from $P_i(u) \cap P_i(v)$ is the same as sampling a color uniformly at random from $P_i(u)$. It follows that

$$\Pr\left[\tilde{\chi}(e) = c\right] \leq \Pr\left[\tilde{\chi}(e) = c \,|\, c \in P_i(u)\right] \leq \frac{1}{|P_i(u)|}.$$

□

A.2.3 Concentration of Basic Quantities

We now establish the concentration of some basic quantities. We first start by analyzing how many edges incident on a node are sampled during a round. For $u \in V$ and $i \in [T]$, let $N_i(u)$ denote the set of edges $N(u) \cap S_i$, and let $N_{\geq i}(u)$ denote the set of edges $\cup_{i' \geq i} N_{i'}(u)$.

Lemma A.9. *For all $u \in V$, $i \in [T]$, we have that*

$$|N_i(u)| < (\epsilon + \epsilon^2)(1-\epsilon)^{i-1}\Delta$$

with probability at least $1 - 1/n^{14}$.

Proof. Let $u \in V$ and $i \in [T]$. Since the round of each edge is sampled from the capped geometric distribution, it follows that

$$\mathbb{E}[|N_i(u)|] = \sum_{e \in N(u)} \Pr[e \in S_i] = \epsilon(1-\epsilon)^{i-1}|N(u)| \leq \epsilon(1-\epsilon)^{i-1}\Delta.$$

Since the rounds of edges are sampled independently, we can apply a Chernoff bound to get concentration. It follows that

$$\Pr\left[|N_i(u)| \geq (1+\epsilon) \cdot \epsilon(1-\epsilon)^{i-1}\Delta\right] \leq \exp(-\epsilon^2 \cdot \epsilon(1-\epsilon)^{i-1}\Delta/3).$$

Now note that

$$(1-\epsilon)^{i-1} \geq e^{-\epsilon T}(1 - \epsilon^2 T) \geq \epsilon(1 - \epsilon\log(1/\epsilon)) \geq \epsilon/2,$$

and hence we have that

$$\exp(-\epsilon^2 \cdot \epsilon(1-\epsilon)^{i-1}\Delta/3) \leq \exp(-\epsilon^4\Delta/6) \leq \exp(-16\log n) = 1/n^{16}.$$

The result follows by union bounding over all $u \in V$, $i \in [T]$, and noting that $T \leq 1/\epsilon^4 \leq n$. □

We now define an event $\mathcal{Z}$ which occurs if and only if $|N_i(u)| < (\epsilon + \epsilon^2)(1-\epsilon)^{i-1}\Delta$ for all $u \in V$, $i \in [T]$. By Lemma A.9, this event occurs with probability at least $1 - 1/n^{14}$. For the rest of Appendix A.2.3, we fix all of the random bits used to determine the rounds of the potential edges and assume that event $\mathcal{Z}$ occurs. We implicitly condition all probabilities on event $\mathcal{Z}$ unless explicitly stated otherwise. Hence, when taking expectations and probabilities, we are doing so over the randomness of the color sequences assigned to edges.

Lemma A.10. *For all $u \in V$, $i \in [T]$, we have that*

$$|P_i(u)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta.$$

Proof. Given any $u \in V$, $i \in [T]$, we have that

$$|N_{<i}(u)| = \sum_{j=1}^{i-1} |N_j(u)| < (\epsilon+\epsilon^2)\Delta \sum_{j=1}^{i-1}(1-\epsilon)^{j-1} = (1+\epsilon)\Delta \cdot (1-(1-\epsilon)^{i-1}).$$

It follows that $|P_i(u)| \geq (1+\epsilon)\Delta - |N_{<i}(u)| \geq (1+\epsilon)(1-\epsilon)^{i-1}\Delta$. $\square$

Lemma A.11. *For all $e \in E$, $i \in [T]$ such that $e \in E_i$ and $e \in N^2(U)$, we have that*

$$|P_i(e)| > (1-\epsilon^2)(1-\epsilon)^{2(i-1)}\Delta$$

with probability at least $1 - 1/n^9$.

Proof. Given any such edge $e = (u, v)$, we have that

$$|P_i(e)| = \sum_{c \in [(1+\epsilon)\Delta]} X_c^u \cdot X_c^v$$

where X_c^w is the indicator for the event that $c \in P_i(w)$ for a color c and a node w. Since one of the endpoints of e is distance at most 2 from a good node, we get that $\mathcal{N}(e, T-1)$ is a tree. Since we only want to analyse the palette $P_i(e)$, we can assume that the input graph is $\mathcal{N}(e, T-1)$, which is a tree. Hence, the connected components containing u and v in the graph $(V, S_{<i})$ are not connected, so by Lemmas A.6 and A.7 it follows that $\{X_c^u\}_c$ and $\{X_c^v\}_c$ are independent families of random variables. Note that, by Lemma A.5, given that $|P_i(u)| = \gamma$ for some $\gamma \in \mathbb{N}$, $\{X_c^u\}_c$ is a permutation distribution where exactly γ of the X_c^u are 1 and the rest are 0. Since we know that $|P_i(v)| > (1+\epsilon)(1-\epsilon)^{i-1}\Delta$, we can define a collection of random variables $\{Y_c^u\}_c$ by taking a uniform random subset Π^u of size $(1+\epsilon)(1-\epsilon)^{i-1}\Delta$ of the set $\{c \in [(1+\epsilon)\Delta] \,|\, X_c^u = 1\}$ and letting Y_c^u indicate whether $c \in \Pi^u$. It follows that $\{Y_c^u\}_c$ is a permutation distribution where exactly $(1+\epsilon)(1-\epsilon)^{i-1}\Delta$ of the Y_u^c are 1, and hence by Proposition E.8 is an NA collection of indicator random variables such that $Y_c^u \leq X_c^u$ for each c. We define $\{Y_c^v\}_c$ in the exact same way. Since $\{X_c^u\}_c$ and $\{X_c^v\}_c$ are independent families, it follows that $\{Y_c^u\}_c$ and $\{Y_c^v\}_c$ are independent families, and we can apply closure under products (Proposition E.9) to get that $\{Y_c^u, Y_c^v\}_c$ is also a family of NA random variables. By then applying disjoint monotone aggregation (Proposition E.9), it follows that $\{Y_c^u \cdot Y_c^v\}_c$ are NA. Hence, we can apply a Chernoff bound. Letting $Y_c = Y_c^u \cdot Y_c^v$, we first note that

$$\mathbb{E}\left[\sum_c Y_c\right] = \sum_c \mathbb{E}\left[Y_c^u\right] \cdot \mathbb{E}\left[Y_c^v\right] = (1+\epsilon)\Delta \cdot \frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)\Delta} \cdot \frac{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)\Delta}$$
$$= (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta.$$

It follows that

$$\Pr\left[\sum_c Y_c \leq (1-\epsilon) \cdot (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta\right] \leq \exp(-\epsilon^2 \cdot (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta/2).$$

As we saw in the proof of Lemma A.9, $(1-\epsilon)^{i-1} \geq \epsilon/2$, so it follows that

$$\exp(-\epsilon^2 \cdot (1+\epsilon)(1-\epsilon)^{2(i-1)}\Delta/2) \leq \exp(-\epsilon^4\Delta/8) \leq \exp(-12\log n) \leq 1/n^{12}.$$

The result follows by union bounding over all $e \in E$, $i \in [T]$, and noting that $|P_i(e)| \geq \sum_c Y_c$. $\square$

A.2.4 Analyzing the Failed Edges

Given some good node $u \in U$, we now bound the number of edges incident on u that fail to be colored, either because the algorithm failed to find a color in its palette or because of conflicts with neighboring edges. For $u \in V$, $i \in [T+1]$, we denote by $F_i(u)$ the set $N_i(u) \cap F_i$ of edges incident on u that fail during iteration i. We begin by bounding the number of edges in $F_{T+1}(u)$, and then proceed to bound $F_i(u)$ for $i \in [T]$. The following lemma is *not* conditioned on event $\mathcal{Z}$.

Lemma A.12. *Let $u \in V$. Then we have that $|F_{T+1}(u)| \leq \epsilon(1+\epsilon)\Delta$ with probability at least $1 - 1/n^{33}$.*

Proof. Let $u \in V$. Given some $e \in N(u)$, we can see that the probability that e is never selected to be colored during a round is $(1-\epsilon)^T$. Since all of these edges are sampled to be colored independently, it follows that

$$\mathbb{E}[|F_{T+1}(u)|] = (1-\epsilon)^T |N(u)| \leq e^{-\epsilon T}\Delta = \epsilon\Delta.$$

By applying Chernoff bounds, it follows that

$$\Pr\left[|F_{T+1}(u)| \geq (1+\epsilon)\cdot\epsilon\Delta\right] \leq \exp(-\epsilon\Delta\cdot\epsilon^2/3) \leq \exp(-33\log n) = 1/n^{33}.$$

□

For the rest of Appendix A.2.4, we again fix all of the random bits used to determine the rounds of the potential edges and assume that event $\mathcal{Z}$ occurs, implicitly conditioning all probabilities on $\mathcal{Z}$.

For some $i \in [T]$, we now categorize the failed edges in $F_i(u)$ into 3 types and bound each of these individually. We say that edges e and f *conflict* if they share an endpoint and are assigned the same tentative color. Note that conflicting edges must receive their tentative colors on the same round. Given some edge $e \in F_i(u)$, we place e into $F_i'(u)$ iff there exists some edge $f \in F_i(u)$ such that e and f conflict and we place e into $F_i''(u)$ iff there exists some edge $f \in F_i \setminus F_i(u)$ such that e and f conflict. $F_i'(u) \cup F_i''(u)$ capture all of the edges that fail due to conflicts, i.e. $F_i'(u) \cup F_i''(u) = \{e \in N_i(u) \mid \exists f \in N_i(e) \text{ such that } \tilde{\chi}(e) = \tilde{\chi}(f)\}$. Finally, we let $F_i'''(u)$ be the edges $e \in F_i(u)$ that fail because the algorithm does not find a color in its palette, i.e. $c_e \cap P_i(e) = \varnothing$. Clearly $F_i(u) = F_i'(u) \cup F_i''(u) \cup F_i'''(u)$. Note that these sets are not necessarily disjoint. We now proceed to bound $|F_i(u)|$ by individually bounding $|F_i'(u)|$, $|F_i''(u)|$ and $|F_i'''(u)|$. We split up the bound in this way because the techniques required to establish concentration on the sizes of these sets are slightly different.

Lemma A.13. *Let $u \in U$, $i \in [T]$. Then we have that $|F_i'(u)| \leq 2(\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at least $1 - 1/n^{25}$.*

Proof. Let $u \in U$, $i \in [T]$, and arbitrarily fix all of the random bits used in the first $i-1$ rounds that determine the palette $P_i(u)$. Let $u_1, \dots, u_\ell$ be the nodes that are connected to u by an edge in $N_i(u)$. Recall that we can assume the input graph is $\mathcal{N}(u, T+1)$, which is a tree. Since the graph is a tree, the nodes $u, u_1, \dots, u_\ell$ are all disconnected from each other in $(V, S_{<i})$, and hence $\{X_{u_1}^c\}_c, \dots, \{X_{u_\ell}^c\}_c$ are mutually independent families of random variables by Lemmas A.6 and A.7.

Let $e \in N_i(u)$. Given some $f \in N_i(u) \setminus \{e\}$, the probability that e and f conflict is the probability that e and f are assigned the same color. Letting $c = \tilde{\chi}(e)$, we have by Lemma A.8 that $\Pr[\tilde{\chi}(f) = c] \leq 1/|P_i(u)|$. Let $f, f' \in N_i(u) \setminus \{e\}$ be distinct edges such that $f = (u, v)$ and $f = (u, v')$. Then the event $\tilde{\chi}(f) = c$ depends on the random bits that determine $P_i(v)$ and the random bits used to sample $\tilde{\chi}(f)$ (recall that the bits that determine $P_i(u)$ are fixed). Hence, by

Lemma A.6, the events $\tilde{\chi}(f) = c$ and $\tilde{\chi}(f') = c$ are independent since they depend on distinct random bits. It follows that

$$\Pr[e \notin F_i'(u)] = \prod_{f \in N_i(u)\setminus\{e\}} \Pr\left[\tilde{\chi}(f) \neq c\right] \geq \prod_{f \in N_i(u)\setminus\{e\}} \left(1 - \frac{1}{|P_i(u)|}\right)$$

$$\geq \left(1 - \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}\right)^{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta} \geq 1 - \frac{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta} = 1 - \epsilon,$$

where the last inequality follows from Bernoulli's inequality. By linearity of expectation, we get that

$$\mathbb{E}[|F_i'(u)|] = \sum_{e \in N_i(u)} \Pr[e \in F_i'(u)] \leq \epsilon |N_i(u)| \leq (\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta.$$

We now establish concentration. Let $N_i(u) = \{e_1, ..., e_\ell\}$ and $\phi'(\tilde{\chi}(e_1), ..., \tilde{\chi}(e_\ell))$ be the function that counts the number of edges in $F_i'(u)$, i.e. $\phi' = |F_i'(u)|$. Since the function ϕ' is Lipschitz with all constants 2 (Definition E.3) and $\tilde{\chi}(e_1), ..., \tilde{\chi}(e_\ell) \in P_i(u)$ are mutually independent as they depend on distinct random bits, we can apply the method of bounded differences (Proposition E.4) to get that

$$\Pr\left[\phi' \geq \mathbb{E}[\phi'] + t\right] \leq \exp\left(-\frac{t^2}{2|N_i(u)|}\right)$$

for all $t > 0$. By setting $t = \epsilon^2(1+\epsilon)(1-\epsilon)^{i-1}\Delta$ we get that $|F_i'(u)| \geq 2(\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at most

$$\exp\left(-\frac{\epsilon^4(1+\epsilon)^2(1-\epsilon)^{2(i-1)}\Delta^2}{2(\epsilon+\epsilon^2)(1-\epsilon)^{i-1}\Delta}\right) = \exp\left(-\frac{\epsilon^3(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{2}\right) \leq \exp\left(-\frac{1}{4}\epsilon^4\Delta\right) \leq \frac{1}{n^{25}}.$$

□

Lemma A.14. *Let $u \in U$, $i \in [T]$. Then we have that $|F_i''(u)| \leq 2(\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at least $1 - 1/n^{33}$.*

Proof. Let $u \in U$, $i \in [T]$, and $u_1, \ldots, u_\ell$ be the nodes that are connected to u by an edge in $N_i(u)$. Arbitrarily fix all of the random bits used in the first $i-1$ rounds that determine the palettes $P_i(u), P_i(u_1), \ldots, P_i(u_\ell)$. Let $v \in \{u_1, \ldots, u_\ell\}$, $e = (u, v)$, and $v_1, \ldots, v_{\ell'}$ be the nodes that are connected to v by an edge in $N_i(u) \setminus \{e\}$. Recall that we can assume the input graph is $\mathcal{N}(u, T+1)$, which is a tree. Since the graph is a tree, the nodes $v, v_1, \ldots, v_\ell$ are all disconnected from each other in $(V, S_{<i})$, and hence $\{X_{v_1}^c\}_c, \ldots, \{X_{v_\ell}^c\}_c$ are mutually independent families of random variables by Lemmas A.6 and A.7.

Given some $f \in N_i(v) \setminus \{e\}$, the probability that e and f conflict is the probability that e and f are assigned the same tentative color. Letting $c = \tilde{\chi}(e)$, we have by Lemma A.8 that $\Pr[\tilde{\chi}(f) = c] \leq 1/|P_i(v)|$. Let $f, f' \in N_i(v) \setminus \{e\}$ be distinct edges such that $f = (v, w)$ and $f = (v, w')$. Then the event $\tilde{\chi}(f) = c$ depends on the random bits that determine $P_i(w)$ and the random bits used to sample $\tilde{\chi}(f)$ (recall that the bits that determine $P_i(v)$ are fixed). Hence, by Lemma A.6, the events $\tilde{\chi}(f) = c$ and $\tilde{\chi}(f') = c$ are independent since they depend on distinct random bits. It follows that

$$\Pr[e \notin F_i''(u)] = \prod_{f \in N_i(v)\setminus\{e\}} \Pr\left[\tilde{\chi}(f) \neq c\right] \geq \prod_{f \in N_i(v)\setminus\{e\}} \left(1 - \frac{1}{|P_i(v)|}\right)$$

$$\geq \left(1 - \frac{1}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta}\right)^{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta} \geq 1 - \frac{\epsilon(1+\epsilon)(1-\epsilon)^{i-1}\Delta}{(1+\epsilon)(1-\epsilon)^{i-1}\Delta} = 1-\epsilon,$$

where the last inequality follows from Bernoulli's inequality. By linearity of expectation, we get that

$$\mathbb{E}[|F_i''(u)|] = \sum_{e \in N_i(u)} \Pr[e \in F_i''(u)] \leq \epsilon |N_i(u)| \leq (\epsilon^2 + \epsilon^3)(1-\epsilon)^{i-1}\Delta.$$

We now establish concentration. Given $e = (u,v), e' = (u,v') \in N_i(u)$, the fact that the events $e \in F_i''(u)$ and $e' \in F_i''(u)$ depend on disjoint collections of random bits follow the details above and the fact that the graph is a tree. Hence, these events are independent. It follows that we can apply a Chernoff bound.

$$\Pr[|F_i''(u)| \geq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta] \leq \exp\left(-\frac{1}{3}(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta\right) \leq \exp\left(-\frac{\epsilon^3}{6}(1+\epsilon)\Delta\right) \leq 1/n^{33}.$$

□

Lemma A.15. *Let $u \in U$, $i \in [T]$. Then we have that $|F_i'''(u)| \leq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta$ with probability at least $1 - 2/n^9$.*

Proof. We begin by fixing all of the random bits used in the first $i-1$ rounds such that for all $e \in N^2(U)$ we have $|P_i(e)| > (1-\epsilon^2)(1-\epsilon)^{2(i-1)}\Delta$. Note that by Lemma A.11 this event occurs with probability at least $1 - 1/n^9$. Given some edge $e \in N_i(u)$, we now have that

$$\Pr[e \in F'''(u)] = \left(1 - \frac{|P_i(e)|}{(1+\epsilon)\Delta}\right)^{8\log(1/\epsilon)/\epsilon^2} \leq \left(1 - (1-\epsilon)(1-\epsilon)^{2(i-1)}\right)^{8\log(1/\epsilon)/\epsilon^2}$$

$$\leq \left(1 - \epsilon^2/8\right)^{8\log(1/\epsilon)/\epsilon^2} \leq \epsilon,$$

where the first equality follows from the fact that we are drawing colors from $[(1+\epsilon)\Delta]$ independently and u.a.r while avoiding $P_i(e)$ and the second inequality follows from the fact that $(1-\epsilon)^{i-1} \geq \epsilon/2$. By linearity of expectation, it follows that

$$\mathbb{E}[|F_i'''(u)|] = \sum_{e \in N_i(u)} \Pr[e \in F_i'''(u)] \leq \epsilon|N_i(u)| < (\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta.$$

Given two distinct edges $e = (u,v), f = (u,w) \in N_i(u)$, the random bits used by the algorithm to sample the colors $\tilde{\chi}(e)$ and $\tilde{\chi}(f)$ are distinct, so we can apply Lemma A.6 to get that the events $e \in F_i'''(u)$ and $f \in F_i'''(u)$ are independent. Hence, we can apply Chernoff bounds to get that

$$\Pr[|F_i'''(u)| \geq 2(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta] \leq \exp\left(-\frac{1}{3}(\epsilon^2+\epsilon^3)(1-\epsilon)^{i-1}\Delta\right) \leq \exp\left(-\frac{\epsilon^3}{6}(1+\epsilon)\Delta\right) \leq 1/n^{33}.$$

□

Let G_F denote the subgraph of G consisting of the edges contained in F. We now remove the conditioning on event $\mathcal{Z}$.

Lemma A.16. *We have that $\deg_{G_F}(u) \leq 7\epsilon(1+\epsilon)\Delta$ for all $u \in U$ with probability at least $1 - 6/n^7$.*

Proof. Let $u \in U$. Then, by Lemmas A.12, A.13, A.14 and A.15 we get that

$$\deg_{G_F}(u) = |F_{T+1}(u)| + \sum_{i=1}^{T} |F_i(u)| \leq |F_{T+1}(u)| + \sum_{i=1}^{T} \left(|F_i'(u)| + |F_i''(u)| + |F_i'''(u)|\right)$$

$$\leq \epsilon(1+\epsilon)\Delta + 6(\epsilon^2+\epsilon^3)\Delta \sum_{i=1}^{T}(1-\epsilon)^{i-1} = \epsilon(1+\epsilon)\Delta + 6(\epsilon^2+\epsilon^3)\Delta \cdot \frac{1-(1-\epsilon)^T}{\epsilon} \leq 7\epsilon(1+\epsilon)\Delta.$$

with probability at least $1 - 1/n^{33} - T/n^{25} - T/n^{33} - 2T/n^9 \geq 1 - 5/n^8$. The lemma follows by union bounding over all nodes in U and removing the conditioning on event $\mathcal{Z}$. $\square$

A.3 Properties of Algorithm 3

Let $G = (V, E)$ be a graph with maximum degree at most Δ and $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ be the subgraphs obtained from calling Algorithm 3 on G with some $2 \leq \Delta' \leq \Delta$.

Lemma A.17. *$\Delta(\mathcal{G}_j) \leq \Delta' + 10\sqrt{\Delta' \log n}$ for all $j \in [\eta]$ with probability at least $1 - 1/n^{31}$.*

Proof. Let $j \in [\eta]$ and $u \in V$. Let X_j^e be the indicator for the event that some edge $e \in E$ is contained in the graph $\mathcal{G}_j$. Then clearly $\deg_{\mathcal{G}_j}(u) = \sum_{e \in N(u)} X_j^e$, and hence

$$\mathbb{E}[\deg_{\mathcal{G}_j}(u)] = \sum_{e \in N_G(u)} \Pr[e \in \mathcal{E}_j] \leq \frac{\Delta}{\eta} \leq \Delta'.$$

By applying Chernoff bounds, we get that

$$\Pr\left[\deg_{\mathcal{G}_j}(u) > \Delta' + 10\sqrt{\Delta' \log n}\right] \leq \exp\left(-\frac{1}{3} \cdot \frac{100 \log n}{\Delta'} \cdot \Delta'\right) \leq \frac{1}{n^{33}},$$

which implies that $\deg_{\mathcal{G}_j}(u) \leq \Delta' + 10\sqrt{\Delta' \log n}$ with probability at least $1 - 1/n^{33}$. We can union bound over all $u \in V$ and $j \in [\eta]$ to get that $\Delta(\mathcal{G}_j) \leq \Delta' + 10\sqrt{\Delta' \log n}$ for all $j \in [\eta]$ with probability at least $1 - n\eta/n^{33} \geq 1 - 1/n^{31}$. $\square$

The following lemma is proven by [KLS+22].

Lemma A.18 (Lemma 4.2, [KLS+22])**.** *Let G' be a subgraph of G obtained by sampling each edge in G independently with probability D/Δ, where $D \geq 2$. Then the probability that the g-neighborhood of an edge e in G' contains a cycle is at most $3D^{5g}/\Delta$.*

We will use the following lemma, which follows immediately from Lemma A.18.

Lemma A.19. *Let G' be a subgraph of G obtained by sampling each edge in G independently with probability D/Δ, where $D \geq 2$. Then the probability that the g-neighborhood of a node u contains a cycle in G' is at most $3D^{5g}/\Delta$.*

Lemma A.20. *Let $G^\star$ be the subgraph of G that contains an edge e iff $e \in \mathcal{E}_j$ is incident on a node u such that the g-neighborhood of e in $\mathcal{G}_j$ is not a tree. Then $\Delta(G^\star) \leq (\Delta' + 10\sqrt{\Delta' \log n}) \cdot (6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta') \log n})$ with probability at least $1 - 1/n^{30}$.*

Proof. Given some $u \in V$, $j \in [\eta]$, define X_j^u to be the indicator for the event that the $(g+1)$-neighborhood of u in the graph $\mathcal{G}_j$ is not a tree. It immediately follows that

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

By Lemma A.17, we have that $\max_j \Delta(\mathcal{G}_j) \leq \Delta' + 10\sqrt{\Delta' \log n}$ with probability at least $1 - 1/n^{31}$. It follows from Lemma A.19 that the $g+1$-neighborhood of u in the graph $\mathcal{G}_j$ is not a tree with probability at most $3(\Delta/\eta)^{5(g+1)}/\Delta \leq 3(\Delta')^{5(g+1)}/\Delta$. Hence, letting X^u denote $\sum_{j \in [\eta]} X_j^u$,

$$\mathbb{E}[X^u] \leq \sum_{j \in [\eta]} \mathbb{E}[X_j^u] \leq \sum_{j \in [\eta]} 3(\Delta')^{5(g+1)}/\Delta \leq 6(\Delta')^{5(g+1)}$$

holds for all $u \in V$. In order to establish concentration, we first establish that, for any fixed u, the random variables $\{X_j^u\}_j$ are NA (see Definition E.5). Given some $e \in E$, let X_j^e indicate the event that $e \in \mathcal{E}_j$. For any fixed j, the random variables $\{X_j^e\}_j$ are NA by Proposition E.6. Since the families of random variables $\{X_j^{e_1}\}_j, \ldots, \{X_j^{e_m}\}_j$ are mutually independent, it follows by closure under products (Proposition E.9) that $\{X_j^e\}_{j,e}$ are NA. Finally, for any $u \in V$, since X_j^u is a monotonically increasing function of $X_j^{e_1}, \ldots, X_j^{e_m}$, it follows by disjoint monotone aggregation (Proposition E.9) that $\{X_j^u\}_j$ are NA. Hence, we can apply Hoeffding bounds for NA random variables to get that

$$\Pr\left[X^u > 6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n}\right] \leq \exp\left(-2 \cdot \frac{100(\Delta/\Delta')\log n}{\eta}\right) \leq \frac{1}{n^{100}}.$$

By union bounding over all $u \in V$, it follows that, with probability at least $1 - 1/n^{99}$,

$$X^u \leq 6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n}$$

for all $u \in V$. Putting everything together and applying a union bound we get that with probability at least $1 - 1/n^{30}$

$$\Delta(G^\star) \leq \left(\Delta' + 10\sqrt{\Delta' \log n}\right) \cdot \left(6(\Delta')^{5(g+1)} + 10\sqrt{(\Delta/\Delta')\log n}\right).$$

$\square$

A.4 Edge Coloring the Subsampled Graphs

Let $G = (V, E)$ be a graph with maximum degree at most Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{30T}$. Let $\gamma = 1/(30T)$ and $\Delta' = \Delta^\gamma$. Now suppose we run Algorithm 2 on input (G, Δ, ϵ). Let $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ denote the partition of G produced by the algorithm. Recall that, for $j \in [\eta]$, $\mathcal{F}_j$ denotes the set of failed edges in $\mathcal{G}_j$. Let $\mathcal{F} = \bigcup_{j \in [\eta]} \mathcal{F}_j$. For notational convenience, we identify $\mathcal{F}$ with the graph $(V, \mathcal{F})$ for the rest of this section. Finally, let $G^\star$ denote the graph that contains an edge $e \in \mathcal{E}_j$ if and only if one of the endpoints of e is bad with respect to $\mathcal{G}_j$. Then we have the total number of colors used by our algorithm is

$$\sum_{j \in [\eta]} (1+\epsilon)^2 \Delta' + 3\Delta(\mathcal{F}),$$

where the factor of 3 in the second term comes from the greedy algorithm. We now proceed to show that, with high probability, this expression is upper bounded by $(1 + O(\epsilon))\Delta$. We begin with the following simple observation.

Observation A.21. *For all $j \in [\eta]$, $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta'$ with probability at least $1 - 1/n^{31}$.*

Proof. By Lemma A.17, we have that for all $j \in [\eta]$

$$\Delta(\mathcal{G}_j) - \Delta' \leq 10\sqrt{\Delta' \log n} \leq 10\epsilon^2\Delta'/(10\sqrt{2}) \leq \epsilon\Delta'$$

with probability at least $1 - 1/n^{31}$. $\square$

For the rest of this section, we assume that the event in the statement of Observation A.21 occurs and implicitly condition all probabilities on this event unless explicitly stated otherwise. Note that we also have $(1+\epsilon)\Delta' \geq (100 \log n)/\epsilon^2$.

Lemma A.22. *We have that $\sum_{j\in[\eta]}(1+\epsilon)^2\Delta' \leq (1+4\epsilon)\Delta$.*

Proof.

$$\sum_{j\in[\eta]}(1+\epsilon)^2\Delta' \leq (1+\epsilon)^2\Delta'\eta \leq (1+\epsilon)^2\Delta + (1+\epsilon)^2 \leq (1+4\epsilon)\Delta.$$

$\square$

Lemma A.23. *Let $u \in V$ and let $J_u^\star = \{j \in [\eta] \,|\, u \text{ is good with respect to } \mathcal{G}_j\}$. Then we have that*

$$\deg_{\mathcal{F}}(u) \leq \Delta(G^\star) + \sum_{j \in J_u^\star} \deg_{\mathcal{F}_j}(u)$$

Proof. We have that

$$\deg_{\mathcal{F}}(u) = \sum_{j\in[\eta]} \deg_{\mathcal{F}_j}(u) = \sum_{j\in J_u^\star} \deg_{\mathcal{F}_j}(u) + \sum_{j\in[\eta]\setminus J_u^\star} \deg_{\mathcal{F}_j}(u).$$

By the definition of the graph $G^\star$, for any $j \in [\eta] \setminus J_u^\star$, all of the edges incident on u in $\mathcal{G}_j$ are contained in $G^\star$. Since the $\mathcal{G}_j$ are edge-disjoint, it follows that

$$\sum_{j\in[\eta]\setminus J_u^\star} \deg_{\mathcal{F}_j}(u) \leq \sum_{j\in[\eta]\setminus J_u^\star} \deg_{\mathcal{G}_j}(u) \leq \deg_{G^\star}(u) \leq \Delta(G^\star).$$

$\square$

Lemma A.24. *For all $j \in [\eta]$, $\deg_{\mathcal{F}_j}(u) \leq 9\epsilon\Delta'$ for all $u \in V$ such that u is good with respect to $\mathcal{G}_j$ with probability at least $1 - 6/n^6$.*

Proof. Let $j \in [\eta]$. It follows from Lemma A.16 that, for all $u \in V$ that are good with respect to $\mathcal{G}_j$, $\deg_{\mathcal{F}_j}(u) \leq 7\epsilon(1+\epsilon)^2\Delta' \leq 9\epsilon\Delta'$ with probability at least $1 - 6/n^7$. The result follows by union bounding over all $j \in [\eta]$. $\square$

Lemma A.25. *We have that $\Delta(G^\star) \leq \epsilon\Delta$ with probability at least $1 - 1/n^{30}$.*

Proof. By Lemma A.20, with probability at least $1 - 1/n^{30}$ we have that

$$\begin{aligned}
\Delta(G^\star) &\leq \left(\Delta^{\gamma} + \Delta^{\gamma/2} \cdot 10\sqrt{\log n}\right) \cdot \left(\Delta^{5\gamma(T+2)} \cdot 6 + \Delta^{(1-\gamma)/2} \cdot 10\sqrt{\log n}\right) \\
&\leq \Delta^{5\gamma T + 11\gamma} \cdot 6 + \Delta^{(1+\gamma)/2} \cdot 10\sqrt{\log n} + \Delta^{5\gamma T+11\gamma} \cdot 60\sqrt{\log n} + \Delta^{1/2} \cdot 100 \log n \\
&\leq 100 \log n \cdot \left(\Delta^{5\gamma T+11\gamma} + \Delta^{(1+\gamma)/2} + \Delta^{1/2}\right) \\
&\leq \left(\Delta^{5/30+11/(30T)} + \Delta^{1/2+1/(60T)}\right) \cdot 200 \log n \\
&\leq \left(\Delta^{5/30+11/(30T)} + \Delta^{1/2+1/(60T)}\right) \cdot 300\epsilon^4 \Delta^{1/(30T)}/200 \\
&\leq 4\epsilon^4 \Delta \\
&\leq \epsilon\Delta.
\end{aligned}$$

□

Lemma A.26. *We have that $\Delta(\mathcal{F}) \leq 19\epsilon\Delta$ with probability at least $1 - 7/n^6$.*

Proof. It follows from the preceding lemmas that for all $u \in V$ we have that

$$\deg_{\mathcal{F}}(u) \leq \Delta(G^\star) + \sum_{j \in J_u^\star} \deg_{\mathcal{F}_j}(u) \leq \epsilon\Delta + |J_u^\star| \cdot 9\epsilon\Delta' \leq 19\epsilon\Delta$$

with probability at least $1 - 1/n^{30} - 6/n^6 \geq 1 - 7/n^6$. It immediately follows that $\Delta(\mathcal{F}) \leq 19\epsilon\Delta$ with the same probability. □

We are not ready to complete the proof of Theorem A.2.

Proof of Theorem A.2. First, observe that our algorithm uses at most

$$\sum_{i \in [\eta]} (1+\epsilon)^2 \Delta' + 3\Delta(\mathcal{F})$$

colors. It now follows from Lemmas A.22 and A.26 that

$$\sum_{i \in [\eta]} (1+\epsilon)^2 \Delta' + 3\Delta(\mathcal{F}) \leq (1+4\epsilon)\Delta + 3 \cdot 19\epsilon\Delta = (1+61\epsilon)\Delta$$

with probability at least $1 - 1/n^{31} - 7/n^6 \geq 1 - 8/n^6$, where the terms in the probability come from removing the conditioning on the event from Observation A.21 and the probability in Lemma A.26. □

B Our Dynamic Algorithm with Constant Recourse (Full Version)

In this appendix, we describe our dynamic algorithm. For now, we omit all details of implementation and deal only with bounding the *recourse* of our dynamic algorithm. We design the data structures that allow us to implement this procedure efficiently in the proceeding sections. The main result in this appendix is the following theorem.

Theorem B.1. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1+61\epsilon)\Delta$-edge coloring of G and has an expected worst-case recourse of $O(1/\epsilon^4)$.*

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

Our algorithm. Informally, our dynamic algorithm works by maintaining the output of StaticColor on the dynamic graph G as it undergoes edge insertions and deletions. In other words, we maintain the invariant that at any point in time the coloring generated by our dynamic algorithm on the current input graph G is the same as the coloring obtained by running StaticColor on G—regardless of what updates have occurred previously. Since we fix the randomness for every potential edge in advance, the tentative colors assigned by StaticColor to the edges in phase 2 (while running Algorithm 4 on the subgraphs $\mathcal{G}_j$) are completely determined by which edges are present in G. Hence, the tentative color $\tilde{\chi}^{(t)}(e)$ of each edge $e \in E^{(t)}$ is well defined. It follows from Theorem A.2 that as long as $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, given some $t \in [\kappa]$, the edge coloring $\chi^{(t)}$ (defined by combining the tentative colorings $\tilde{\chi}^{(t)}$ for the edges in each $\mathcal{G}_j$ that don't fail and the greedy coloring of H) uses at most $(1+61\epsilon)\Delta$ colors with probability at least $1 - O(1/n^6)$.

We design data structures that allow us to maintain these tentative colorings explicitly, along with the sets of edges that fail, for each graph $\mathcal{G}_j$ as edges are inserted and deleted from G. We then dynamically maintain a greedy edge coloring of the graph H that uses $3\Delta(H)$ many colors and only changes the colors of $O(1)$ many edges every time an edge is inserted or deleted from H. This level of detail is sufficient to upper bound the recourse of our dynamic algorithm. We begin by bounding the recourse of our algorithm and defer the details of how to implement this efficiently to the proceeding sections.

B.1 Recourse analysis

We now proceed to upper bound the expected recourse of our algorithm. We achieve this by arguing that in expectation the tentative colorings produced by StaticColor on two graphs that differ only by one edge assign different colors to at most $O(1/\epsilon^4)$ many edges. By showing that the number of edges that are added and removed from H is at most a constant factor larger than the number of edges that change their tentative colors, it then follows that the expected recourse of our algorithm is $O(1/\epsilon^4)$.

Let $t \in [\kappa]$. Suppose that the t^{th} update corresponds to the insertion or deletion of an edge e contained in $\mathcal{G}_j$. Then clearly only edges contained in $\mathcal{G}_j$ can change their tentative colors during this update. Since the colors used by the tentative colorings in each of $\mathcal{G}_1, \ldots, \mathcal{G}_\eta$ are distinct, it also follows that only edges in $\mathcal{G}_j$ can be added or removed from H during the update. It follows that while bounding the recourse of the t^{th} update it is sufficient to only consider the edges contained in the graph $\mathcal{G}_j$. It is not too difficult to see that the recourse of the t^{th} update is upper bounded by

$$O\left(\left|\mathcal{F}_j^{(t-1)} \oplus \mathcal{F}_j^{(t)}\right|\right) + \left|\left\{e \in \mathcal{E}_j^{(t)} \,\middle|\, \tilde{\chi}^{(t-1)}(e) \neq \tilde{\chi}^{(t)}(e)\right\}\right|,$$

where the terms correspond to the changes in the greedy coloring caused by edge insertions and deletions from H due to edges that failed at time $t-1$ but not at time t and vice versa, and the edges that change their tentative colors, respectively.

Notation. In order to emphasize that we are looking at the state of an object X *directly after the t^{th} update* we add the superscript $X^{(t)}$. For example, $P_i^{(t)}(e)$ is the set $P_i(e)$ after the t^{th} update, i.e. when the input graph to our algorithm is $G^{(t)}$, where $P_i(e)$ is the palette of the edge e during round i as defined in Appendix A. Since while bounding the recourse of the t^{th} update it is sufficient to only consider the edges contained in the graph $\mathcal{G}_j$, when we use notation from Algorithm 4, it is implicit that the notation is referring to objects from the call to this algorithm on the graph $\mathcal{G}_j$. We will now introduce some definitions that will allow us to analyze the way that changes in the tentative coloring propagate through the rounds of Algorithm 4 after an update.

The sets $\Gamma(e)$ and $\Lambda(e)$. The following definitions capture the notion of an edge e having an effect on the tentative color assigned to some edge f in a subsequent round and are crucial for efficiently identifying the edges whose tentative colors change after an update.

Definition B.2. *Given some $t \in [\kappa]$, $i \in [T]$, and an edge $e \in S_i^{(t-1)} \cup S_i^{(t)}$, we define the sets of edges $\Gamma^{(t)}(e)$ and $\Lambda^{(t)}(e)$ by*

$$\Gamma^{(t)}(e) := \left\{f \in N_{>i}^{(t)}(e) \,\middle|\, \tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e)\right\},$$

$$\Lambda^{(t)}(e) := \left\{f \in N_{>i}^{(t)}(e) \,\middle|\, \tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t-1)}(e)\right\}.$$

We denote the sets $S_i^{(t)} \cap \Gamma^{(t)}(e)$ and $S_i^{(t)} \cap \Lambda^{(t)}(e)$ by $\Gamma_i^{(t)}(e)$ and $\Lambda_i^{(t)}(e)$ respectively. Informally, one can think of the edges in $\Gamma^{(t)}(e)$ as the edges that change their color during the t^{th} update because e now occupies their color, and the edges in $\Lambda^{(t)}(e)$ as the edges that change their color during the t^{th} update because e no longer occupies a color that they would rather have assigned to them.

Type A and B dirty edges. For some $t \in [\kappa]$, $t \in [T]$, we define the sets of edges $A_i^{(t)}$ and $B_i^{(t)}$ by

$$A_i^{(t)} := \left\{e \in S_i^{(t-1)} \cup S_i^{(t)} \,\middle|\, \tilde{\chi}^{(t-1)}(e) \neq \tilde{\chi}^{(t)}(e)\right\},$$

$$B_i^{(t)} := F_i^{(t-1)} \oplus F_i^{(t)}.$$

In words, $A_i^{(t)}$ is the set of edges in round i that change their tentative color during the t^{th} update, and $B_i^{(t)}$ is the set of edges in round i that fail at either time $t-1$ or t, but not both. We let $A^{(t)}$ and $B^{(t)}$ denote the sets $\bigcup_i A_i^{(t)}$ and $\bigcup_i B_i^{(t)}$ respectively, and refer to the edges in $A^{(t)}$ and $B^{(t)}$ as A-dirty and B-dirty respectively. We define an edge e as being *dirty* with respect to the t^{th} update if the color $\chi(e)$ changes during the t^{th} update and denote the set of such edges by $D^{(t)}$, and $D^{(t)} \cap S_i^{(t)}$ by $D_i^{(t)}$. The recourse of the t^{th} update is precisely $|D^{(t)}|$.

B.1.1 Basic Facts

We now give some basic facts about these definitions. Let $t \in [\kappa]$, $e^\star$ be the edge that is either inserted or deleted during the t^{th} update, and let $i^\star$ denote $i_{e^\star}$.

Lemma B.3. *We have that* $|D^{(t)}| \leq O(|B^{(t)}|) + |A^{(t)}|$.

Proof. It is sufficient to argue that at most $O(|B^{(t)}|)$ many edges not in $A^{(t)}$ change their colors during the t^{th} update. Clearly, any such edge must be contained in $F^{(t-1)} \cup F^{(t)}$. Since at most $O(|F^{(t-1)} \oplus F^{(t)}|)$ many edges in $F^{(t-1)} \cup F^{(t)}$ change their colors during the t^{th} update (recall that our dynamic greedy algorithm changes the colors of at most $O(1)$ many edges in H when adding or removing an edge from H) the lemma follows. $\square$

Lemma B.4. *We have that* $|B^{(t)}| \leq 4|A^{(t)}| + 1$.

Proof. Consider the set of all the failed edges

$$F^{(t)} = \left\{ e \in E^{(t)} \,\middle|\, \exists f \in N^{(t)}_{i_e}(e) \text{ such that } \tilde{\chi}^{(t)}(e) = \tilde{\chi}^{(t)}(f) \right\} \cup \left\{ e \in E^{(t)} \,\middle|\, \tilde{\chi}^{(t)}(e) = \perp \right\}$$

that is defined by our tentative coloring $\tilde{\chi}^{(t)}$. We want to get an upper bound on the size of $B^{(t)} = F^{(t-1)} \oplus F^{(t)}$. Suppose we start with the set $F^{(t-1)}$ and are given the set of edges that change their tentative colors during the t^{th} update, $A^{(t)}$. Let $A^{(t)} = \{e_1, \ldots, e_\ell\}$. Now suppose we update the tentative colors of the first r edges $e_1, \ldots, e_r \in A^{(t)}$, and let $F(r)$ be the set of failed edges defined with respect to the tentative coloring where an edge $f \in E^{(t-1)} \cup E^{(t)} \setminus \{e_1, \ldots, e_r\}$ receives color $\tilde{\chi}^{(t-1)}(f)$ and the edges $e_1, \ldots, e_r$ receive colors $\tilde{\chi}^{(t)}(e_1), \ldots, \tilde{\chi}^{(t)}(e_r)$ respectively. We can see that

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

Let $r \in [\ell]$ and let $e_r = (u, v)$, and consider how $F(r-1)$ changes into $F(r)$ after we change the tentative color of e_r from $\tilde{\chi}^{(t-1)}(e_r)$ to $\tilde{\chi}^{(t)}(e_r)$ (assume for now that neither color is $\perp$). We have that $|F(r) \setminus F(r-1)| \leq 2$. This is because the only edges in $F(r) \setminus F(r-1)$ are edges incident to u and v with color $\tilde{\chi}^{(t)}(e_r)$ that are not already in $F(r-1)$, and there can be at most one such edge incident on each of u and v. By an analogous argument, $|F(r-1) \setminus F(r)| \leq 2$. It follows that $|F(r-1) \oplus F(r)| \leq 4$. A similar argument shows that if one of these colors is $\perp$ then $|F(r-1) \oplus F(r)| \leq 3$. Finally, we note that $F^{(t-1)} \oplus F(0) \subseteq \{e^\star\}$ and the lemma follows. $\square$

Corollary B.5. *We have that* $|D^{(t)}| \leq O(|A^{(t)}|) + O(1)$.

Proof. Follows immediately from Lemmas B.3 and B.4. $\square$

Lemma B.6. *For all* $e \notin A^{(t)}$, *we have that* $\Gamma^{(t)}(e) = \varnothing$ *and* $\Lambda^{(t)}(e) = \varnothing$.

Proof. Since $e \notin A^{(t)}$, we have that $\tilde{\chi}^{(t)}(e) = \tilde{\chi}^{(t-1)}(e)$. Hence, $\Gamma^{(t)}(e) = \{f \in N^{(t)}_{<i_e}(e) \,|\, \tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t-1)}(e)\}$ which is clearly empty since, for any $f \in \Gamma^{(t)}(e)$, e and f share an endpoint and $i_e < i_f$, and hence cannot have the same tentative color. Similarly, the set $\Lambda^{(t)}(e)$ is $\{f \in N^{(t)}_{>i_e}(e) \,|\, \tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t)}(e)\}$ and is also empty by the same arguments. $\square$

Lemma B.7. *For all* $e \in E^{(t-1)} \cup E^{(t)}$, *we have that* $\Gamma^{(t)}(e) \subseteq A^{(t)}$ *and* $\Lambda^{(t)}(e) \subseteq A^{(t)}$.

Proof. Let $e \in E^{(t)} \cup E^{(t-1)}$ and $f \in \Gamma^{(t)}(e)$. Then we know that $\tilde{\chi}^{(t)}(e) = \tilde{\chi}^{(t-1)}(f)$. Since e and f share an endpoint and $i_e < i_f$, we also have that $\tilde{\chi}^{(t)}(e) \neq \tilde{\chi}^{(t)}(f)$. Hence, it follows that $\tilde{\chi}^{(t)}(f) \neq \tilde{\chi}^{(t-1)}(f)$ and so $f \in A^{(t)}$. It follows that $\Gamma^{(t)}(e) \subseteq A^{(t)}$. By an analogous argument, we have that $\Lambda^{(t)}(e) \subseteq A^{(t)}$. $\square$

Lemma B.8. *For all i such that $i^\star < i \le T$, we have that $A_i^{(t)} \subseteq \Gamma^{(t)}(A_{<i}^{(t)}) \cup \Lambda^{(t)}(A_{<i}^{(t)})$.*

Proof. We prove this lemma by showing that $\{e \in A_i^{(t)} \mid \ell_e^{(t)} > \ell_e^{(t-1)}\} \subseteq \Gamma^{(t)}(A_{<i}^{(t)})$ and $\{e \in A_i^{(t)} \mid \ell_e^{(t)} < \ell_e^{(t-1)}\} \subseteq \Lambda^{(t)}(A_{<i}^{(t)})$, which implies that

$$\begin{aligned} A_i^{(t)} &= \{e \in A_i^{(t)} \mid \ell_e^{(t)} > \ell_e^{(t-1)}\} \sqcup \{e \in A_i^{(t)} \mid \ell_e^{(t)} < \ell_e^{(t-1)}\} \\ &\subseteq \Gamma^{(t)}(A_{<i}^{(t)}) \cup \Lambda^{(t)}(A_{<i}^{(t)}). \end{aligned}$$

Let $e \in A_i^{(t)}$ such that $\ell_e^{(t)} > \ell_e^{(t-1)}$ and $i = i_e$. Then there exists some $f \in N^{(t)}(e) \cap S_{<i}^{(t)}$ such that $\tilde{\chi}^{(t)}(f) = c_e(\ell_e^{(t-1)}) = \tilde{\chi}^{(t-1)}(e)$. Hence, $e \in \Gamma^{(t)}(f)$. Since e and f share an endpoint and $i_f < i_e$, we must have that $\tilde{\chi}^{(t-1)}(f) \neq \tilde{\chi}^{(t-1)}(e) = \tilde{\chi}^{(t)}(f)$ and so $f \in A_{<i}^{(t)}$. It follows that $e \in \Gamma^{(t)}(A_{<i}^{(t)})$ and hence $\{e \in A_i^{(t)} \mid \ell_e^{(t)} > \ell_e^{(t-1)}\} \subseteq \Gamma^{(t)}(A_{<i}^{(t)})$. By a similar argument, we get that $\{e \in A_i^{(t)} \mid \ell_e^{(t)} < \ell_e^{(t-1)}\} \subseteq \Lambda^{(t)}(A_{<i}^{(t)})$. $\square$

Corollary B.9. *For all i such that $i^\star < i \le T$, we have that $A_i^{(t)} = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)})$.*

Proof. It follows by Lemmas B.7 and B.8 that $A_i^{(t)} \subseteq \Gamma^{(t)}(A_{<i}^{(t)}) \cup \Lambda^{(t)}(A_{<i}^{(t)}) \subseteq A^{(t)}$. By intersecting with $S_i^{(t)}$, it follows that $A_i^{(t)} \subseteq \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)}) \subseteq A_i^{(t)}$. $\square$

B.1.2 Bounding the Expected Recourse

We can now bound the expected recourse of our dynamic algorithm by showing that $\mathbb{E}[|A^{(t)}|] = O(1/\epsilon^4)$ for all $t \in [\kappa]$. By then applying Corollary B.5, this immediately implies that the expected recourse of our algorithm while handling an update is $O(1/\epsilon^4)$. We devote the rest of this section to proving the following lemma.

Lemma B.10. *For all $t \in [\kappa]$, $\mathbb{E}\left[\left|A^{(t)}\right|\right] \le 1/\epsilon^4 + o(1)$.*

B.2 Proof of Lemma B.10

For the remainder of this section, we fix some $t \in [\kappa]$. Let $e^\star$ denote the edge that is either inserted or deleted during the t^{th} update and let $i^\star$ denote $i_{e^\star}$. We can make the following observation.

Observation B.11. $A_i^{(t)} = \varnothing$ *for all* $0 \le i < i^\star$ *and* $A_{i^\star}^{(t)} \subseteq \{e^\star\}$.

We now establish a relationship between the expected sizes of the sets $A_i^{(t)}$ when i is larger than $i^\star$, and use this to bound the expected size of $A^{(t)}$.

Let $\mathcal{E}$ be the event that $\mathcal{N}(e^\star, 2T+2)$ is a tree at time t and $t-1$. Recall that since we are only concerned with bounding the recourse at time t, we only consider the edges in the graph $\mathcal{G}_j$, where $e^\star$ is contained in $\mathcal{G}_j$. Hence, $\mathcal{N}(e^\star, 2T+2)$ denotes $\mathcal{N}_{\mathcal{G}_j}(e^\star, 2T+2)$ in this context. Let $\mathcal{Z}^{(t)}$ denote the event that $\mathcal{Z}$ (defined in Appendix A.2.3) occurs at time t. Let $\mathcal{Y}^{(t)}$ denote the event that the statement in Observation A.21 occurs at time t. We first argue that we can assume that the event $\mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$ occurs.

Lemma B.12. *We have that*

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \mid \mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}] + o(1).$$

Proof. By the law of total expectation, we have that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + \mathbb{E}[|A^{(t)}| \mid \neg\mathcal{E}] \cdot \Pr[\neg\mathcal{E}].$$

By Lemma A.18, the event $\mathcal{E}$ does not occur with probability at most $3(\Delta')^{10T+10}/\Delta = 3\Delta^{1/(3T)-2/3}$. By Lemma A.3, we can see that every edge in $A^{(t)}$ is contained in $\mathcal{N}(e^\star, T+1)$. Hence, the number of edges contained in $\mathcal{N}(e^\star, T+1)$ is an upper bound on $|A^{(t)}|$. By Observation A.21, we have that $\Delta(\mathcal{G}_j) \leq (1+\epsilon)\Delta' = (1+\epsilon)\Delta^{1/(30T)}$ with probability at least $1 - 1/n^{31}$. It follows that $\mathcal{N}(e^\star, T+1)$ contains at most $2((1+\epsilon)\Delta^{1/(30T)})^{T+1} \leq (3/\epsilon)\Delta^{1/15}$ many edges with the same probability. Putting everything together, we get that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + \mathbb{E}[|A^{(t)}| \mid \neg\mathcal{E}] \cdot \Pr[\neg\mathcal{E}] \leq \mathbb{E}[|A^{(t)}| \mid \mathcal{E}] + 9\Delta^{-4/15}/\epsilon^2$$

with probability at least $1 - 1/n^{31}$. Noting that $|A^{(t)}| \leq |E| \leq n^2$, we have that

$$\mathbb{E}[|A^{(t)}|] \leq n^2 \cdot (1/n^{31}) + \mathbb{E}[|A^{(t)}| \mid \mathcal{E}] + 9\Delta^{-4/15}/\epsilon^2 \leq \mathbb{E}[|A^{(t)}| \mid \mathcal{E}] + 9\Delta^{-4/15}/\epsilon^2 + 1/n^{29}.$$

Finally, letting $\mathcal{X} = \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$, the result follows by noting that

$$\begin{aligned}\mathbb{E}[|A^{(t)}| \mid \mathcal{E}] &\leq \mathbb{E}[|A^{(t)}| \mid \mathcal{E} \cap \mathcal{X}] + n^2 \cdot \Pr[\neg\mathcal{X}] \\ &\leq \mathbb{E}[|A^{(t)}| \mid \mathcal{E} \cap \mathcal{X}] + 4/n^{12}.\end{aligned}$$

□

For the rest of this section, we assume that $\mathcal{N}(e^\star, 2T+2)$ is a tree at time t and $t-1$. We also fix the random bits used by the algorithm to determine the partition of the graph so that event $\mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$ occurs and the random bits used by the algorithm to determine the rounds of edges so that the event $\mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)}$ occurs. Note that, in this context, the upper bound on $\Delta(\mathcal{G}_j)$ is $(1+\epsilon)\Delta'$, where $\Delta' = \Delta^{1/(30T)}$. We implicitly condition all probabilities on these events unless stated otherwise. By Lemma A.3, since all edges that change their tentative color or failed status during the t^{th} update are contained in $\mathcal{N}^{(t)}(e^\star, T+1)$, we can assume that the input graph is $\mathcal{N}^{(t)}(e^\star, 2T+2)$ while trying to bound the number of such edges.

Lemma B.13. *Let $i \in [T]$ and $e \in S_{<i}^{(t)}$, then we have that $\mathbb{E}[|\Gamma_i^{(t)}(e)|] \leq 2\epsilon$.*

Proof. We begin by observing that by linearity of expectation

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

If $e \notin A_{<i}^{(t)}$, then by Lemma B.6 we have that $\Gamma^{(t)}(e) = \varnothing$. Assume that $e \in A_{<i}^{(t)}$. Then e is contained in $\mathcal{N}(e^\star, T+1)$ and hence $\mathcal{N}(e, T+1) \subseteq \mathcal{N}(e^\star, 2T+2)$ is a tree. Let $e = (u, v)$ and fix the random bits used by the algorithm in the first $i-1$ rounds that determine the palettes $P_i^{(t-1)}(u)$ and $P_i^{(t-1)}(v)$. Let $f = (u, w) \in N_i^{(t-1)}(e)$ (note that $N_i^{(t)}(e) = N_i^{(t-1)}(e)$ since $i_{e^\star} < i$). Then e and w are disconnected in the graphs $(V, S_{<i}^{(t-1)})$ and $(V, S_{<i}^{(t)})$ since $\mathcal{N}^{(t)}(e^\star, 2T+2)$ and $\mathcal{N}^{(t-1)}(e^\star, 2T+2)$ are both trees and one is a subgraph of the other. Letting $c = \tilde{\chi}^{(t)}(e)$, we have that c does not

depend on the random bits that determine the palette $P_i^{(t-1)}(w)$. We can now apply Lemmas A.8 and A.10 to get that

$$\Pr[\tilde{\chi}^{(t-1)}(f) = c] \leq \frac{1}{|P_i^{(t-1)}(u)|} \leq \frac{1}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}.$$

The same holds for all $f \in N_i^{(t-1)}(v)$. It follows that

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

$$\leq \sum_{f \in N_i^{(t-1)}(u)} \Pr[\tilde{\chi}^{(t-1)}(f) = c] + \sum_{f \in N_i^{(t-1)}(v)} \Pr[\tilde{\chi}^{(t-1)}(f) = c]$$

$$\leq \frac{|N_i^{(t-1)}(u)| + |N_i^{(t-1)}(v)|}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} \leq \frac{2\epsilon(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} = 2\epsilon.$$

□

Lemma B.14. *Let $i \in [T]$ and $e \in S_{<i}^{(t)}$, then we have that $\mathbb{E}[|\Lambda_i^{(t)}(e)|] \leq 2\epsilon$.*

Proof. We begin by observing that by linearity of expectation

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

If $e \notin A_{<i}^{(t)}$, then by Lemma B.6 we have that $\Lambda^{(t)}(e) = \varnothing$. Assume that $e \in A_{<i}^{(t)}$. Then e is contained in $\mathcal{N}(e^\star, T+1)$ and hence $\mathcal{N}(e, T+1) \subseteq \mathcal{N}(e^\star, 2T+2)$ is a tree. Let $e = (u, v)$ and fix the random bits used by the algorithm in the first $i-1$ rounds that determine the palettes $P_i^{(t)}(u)$ and $P_i^{(t)}(v)$. Let $f = (u, w) \in N_i^{(t)}(e)$. Then e and w are disconnected in the graphs $(V, S_{<i}^{(t-1)})$ and $(V, S_{<i}^{(t)})$ since $\mathcal{N}^{(t)}(e^\star, 2T+2)$ and $\mathcal{N}^{(t-1)}(e^\star, 2T+2)$ are both trees and one is a subgraph of the other. Letting $c = \tilde{\chi}^{(t)}(e)$, we have that c does not depend on the random bits that determine the palette $P_i^{(t)}(w)$. We can now apply Lemmas A.8 and A.10 to get that

$$\Pr[\tilde{\chi}^{(t)}(f) = c] \leq \frac{1}{|P_i^{(t)}(u)|} \leq \frac{1}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}.$$

The same holds for all $f \in N_i^{(t)}(v)$. It follows that

$$\mathbb{E}[|\Lambda_i^{(t)}(e)|] = \sum_{f \in N_i^{(t)}(e)} \Pr[\tilde{\chi}^{(t)}(f) = c]$$

$$\leq \sum_{f \in N_i^{(t)}(u)} \Pr[\tilde{\chi}^{(t)}(f) = c] + \sum_{f \in N_i^{(t)}(v)} \Pr[\tilde{\chi}^{(t)}(f) = c]$$

$$\leq \frac{|N_i^{(t)}(u)| + |N_i^{(t)}(v)|}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} \leq \frac{2\epsilon(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'}{(1+\epsilon)^2(1-\epsilon)^{i-1}\Delta'} = 2\epsilon.$$

□

Lemma B.15. *For all i such that $i^\star < i \leq T$, we have that $\mathbb{E}[|A_i^{(t)}|] \leq 4\epsilon \cdot \mathbb{E}[|A_{<i}^{(t)}|]$.*

Proof. We know by Corollary B.9 that

$$A_i^{(t)} = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)}) = \bigcup_{e \in A_{<i}^{(t)}} \left(\Gamma_i^{(t)}(e) \cup \Lambda_i^{(t)}(e) \right).$$

From this, we can immediately deduce that

$$\mathbb{E}[|A_i^{(t)}|] \leq \sum_{e \in A_{<i}^{(t)}} \left(\mathbb{E}[|\Gamma_i^{(t)}(e)|] + \mathbb{E}[|\Lambda_i^{(t)}(e)|] \right)$$

by using linearity of expectation. It then follows from Lemmas B.13 and B.14 that $\mathbb{E}[|A_i^{(t)}|] \leq 4\epsilon \cdot |A_{<i}^{(t)}|$. The lemma follows by taking expectations on both sides. □

Lemma B.16. *For all i such that $i^\star < i \leq T$, we have that $\mathbb{E}[|A_{\leq i}^{(t)}|] \leq (1 + 4\epsilon) \cdot \mathbb{E}[|A_{\leq i-1}^{(t)}|]$.*

Proof. By applying Lemma B.15 we get that

$$\mathbb{E}[|A_{\leq i}^{(t)}|] = \mathbb{E}[|A_i^{(t)}|] + \mathbb{E}[|A_{<i}^{(t)}|] \leq 4\epsilon \cdot \mathbb{E}[|A_{<i}^{(t)}|] + \mathbb{E}[|A_{<i}^{(t)}|] = (1 + 4\epsilon) \cdot \mathbb{E}[|A_{\leq i-1}^{(t)}|].$$

□

Lemma B.17. *We have that $\mathbb{E}[|A^{(t)}|] \leq 1/\epsilon^4$.*

Proof. It follows from Observation B.11 and Lemma B.16 that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A_{\leq T}^{(t)}|] \leq (1 + 4\epsilon)^{T - i^\star} \cdot \mathbb{E}[|A_{\leq i^\star}^{(t)}|] \leq (1 + 4\epsilon)^T \leq e^{4 \log(1/\epsilon)} = 1/\epsilon^4.$$

□

Finally, using Lemma B.12, we remove the conditioning on the event $\mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)} \cap \mathcal{Y}^{(t-1)} \cap \mathcal{Y}^{(t)}$ to get that

$$\mathbb{E}[|A^{(t)}|] = \mathbb{E}[|A^{(t)}| \mid \mathcal{E} \cap \mathcal{Z}^{(t-1)} \cap \mathcal{Z}^{(t)}] + o(1) = O(1/\epsilon^4).$$

C Implementing our Dynamic Algorithm (Full Version)

We now proceed to give a more detailed description of our algorithm, which we then implement with appropriate data structures. The main result in this appendix is Corollary C.5, which is restated below.

Theorem C.1. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1 + 61\epsilon)\Delta$-edge coloring of G and has an expected worst-case update time of $O(\log^4(1/\epsilon)/\epsilon^9)$.*

C.1 Our Algorithm

High-level approach. Let $t \in [\kappa]$, and suppose we have the edge coloring $\chi^{(t-1)}$ of $G^{(t-1)}$. Let $e^\star$ be the edge inserted/deleted at time t and let $i^\star$ denote $i_{e^\star}$. We know that the edge $e^\star$ is A-dirty (as long as it doesn't fail to be assigned a tentative color) and that there are no other A-dirty edges in the first $i^\star$ rounds. The main idea behind our dynamic algorithm is to iterate through the rounds $i^\star+1, \dots, T$ and observe how the changes in the tentative coloring $\tilde{\chi}$ propagate through the rounds, finding all of the A-dirty edges. By designing an appropriate data structure that updates the failed edges as we update the tentative colors of edges, we show that we can explicitly maintain the failed edges by just finding the A-dirty edges and updating their tentative colors. We find all of the A-dirty edges in round i inductively by using Corollary B.9. In other words, if we can find all of the A-dirty edges in the first $i-1$ rounds then we can find all of the A-dirty edges in round i by using the fact that $A_i^{(t)} = \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)})$. Algorithm 5 gives the pseudocode for this high-level approach.

Algorithm 5 UPDATE-COLORING($e^\star$)

Find $\tilde{\chi}^{(t)}(e^\star)$
if $\tilde{\chi}^{(t)}(e^\star) = \tilde{\chi}^{(t-1)}(e^\star)$ **then**
  return
$A_{i^\star}^{(t)} \leftarrow \{e^\star\}$
for $i = i^\star + 1, \dots, T$ **do**
  $A_i^{(t)} \leftarrow \Gamma_i^{(t)}(A_{<i}^{(t)}) \cup \Lambda_i^{(t)}(A_{<i}^{(t)})$
  Find $\tilde{\chi}^{(t)}(e)$ for all $e \in A_i^{(t)}$

Maintaining the tentative colorings. The main technical challenge that we face is maintaining the tentative colorings for the graphs $\mathcal{G}_j$. In Section C.2, we design a data structure that, given a dynamic graph G', is capable of maintaining and updating a tentative coloring of G' so that the output matches that of Algorithm 4 when they use the same random bits. More precisely, our dynamic data structure will be given the round i_e and color sequence c_e for each edge e in G', and will maintain the color indices ℓ_e of each edge so that they match the corresponding indices produced by Algorithm 4 when run on input G'. This defines a tentative coloring (by setting $\tilde{\chi}(e) = c_e(\ell_e)$) and a corresponding set of failed edges, which the data structure maintains explicitly. Given an edge e to be inserted or deleted from G', our data structure is capable of efficiently identifying all of the edges that need to change their color indices in order for the values maintained by the data structure to match the output produced by running Algorithm 4 on the updated graph. We will create η copies $\mathcal{D}_1, \dots, \mathcal{D}_\eta$ of this data structure and use them to maintain the tentative colorings of the graphs $\mathcal{G}_1, \dots, \mathcal{G}_\eta$ by feeding the graph $\mathcal{G}_j$ to $\mathcal{D}_j$. Since each $\mathcal{D}_j$ explicitly maintains the edges that fail while tentatively coloring $\mathcal{G}_j$, we can use them to explicitly maintain the graph H consisting of all failed edges by adding or removing an edge e from H every time one of the $\mathcal{D}_j$ adds or removes e from its collection of failed edges. In Section C.3, we prove the following lemma.

Lemma C.2. *There exists a dynamic data structure that, given a dynamic graph G and a parameter Δ, is capable of explicitly maintaining the tentative coloring $\tilde{\chi}$ and the set of failed edges F defined by running Algorithm 4 on G, and has an expected update time of $O(\log^4(1/\epsilon)/\epsilon^5 \cdot \mathbb{E}[|A|])$, where A is the set of A-dirty edges during an update.*

Maintaining the greedy coloring. We now show that there exists an algorithm that can dynamically maintain a greedy coloring of a graph and has $O(1)$ expected update time. Our algorithm has the property that each edge insertion or deletion can only change the colors of at most $O(1)$ many edges in the graph. Furthermore, this coloring is maintained explicitly and hence can be queried in $O(1)$ time. More precisely, in Section C.4, we prove the following lemma.

Lemma C.3. *There exists a dynamic data structure that, given a dynamic graph $G = (V, E)$ that undergoes edge insertions and deletions, can explicitly maintain a $3\Delta(G)$-edge coloring of G, where $\Delta(G)$ is the current maximum degree of G, has an expected update time of $O(1)$, and only changes the colors of $O(1)$ many edges per update.*

Putting everything together. We can now use the dynamic algorithm that is described in Lemma C.2 to maintain the tentative coloring $\tilde{\chi}$ and the graph H defined by running Algorithm 2 on G, and then run the dynamic greedy algorithm described in Lemma C.3 on H in order to efficiently maintain a $3\Delta(H)$-edge coloring of H. Recall that we create η copies of this data structure and use them to maintain the tentative colorings and failed edges of each $\mathcal{G}_j$ individually. By Lemma B.10, we know that the expected type A recourse (in the subsampled graph $\mathcal{G}_j$ that the update occurs) is $O(1/\epsilon^4)$, and hence the expected update time of our algorithm is $O(\log^4(1/\epsilon)/\epsilon^9)$. Since the graph H is maintained explicitly, we insert or delete at most $O(\log^4(1/\epsilon)/\epsilon^9)$ many edges from H. Since our dynamic greedy algorithm has an expected update time of $O(1)$, this only leads to a $O(1)$ multiplicative factor in overhead to the update time. Finally, we note that the assumption that the randomness is generated in advance was made purely for analytic purposes so that we could argue that our dynamic algorithm and our static algorithm produce the same output when using the same random bits. However, if we generate the random bits for an edge e on the fly when it is inserted into the graph (i.e. decide which graph $\mathcal{G}_j$ to place it into and sample its round i_e and color sequence c_e independently of all previous random processes) then the distribution of the random bits assigned to the edges in the graph at any given time does not change, and the same guarantees hold. Our main theorem follows.

Theorem C.4. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most Δ at all times, maintains a $(1 + 61\epsilon)\Delta$-edge coloring of G with probability at least $1 - 8/n^6$ at each time $t \in [\kappa]$, and has an expected worst-case update time of $O(\log^4(1/\epsilon)/\epsilon^9)$.*

By our proof of Theorem A.2, it follows that, at any point in time, the graph H has a maximum degree of at most $19\epsilon\Delta$ with probability at least $1-8/n^6$. Hence, in the event that $\Delta(H) > 19\epsilon\Delta$, we can resample all of the randomness used by our algorithm by deleting and reinserting every edge (and resampling all of the random bits for each edge in the process). This will take $O_\epsilon(m)$ expected time, and we will have that $\Delta(H) > 19\epsilon\Delta$ with probability at most $8/n^6$ independently of the randomness that our algorithm had used before. We repeat this process of resampling all of the randomness until we have that $\Delta(H) \leq 19\epsilon\Delta$. In expectation, we do this at most $1/(1 - 8/n^6) - 1 = 1/\Omega(n^6)$ many times. Since we spend $O_\epsilon(m) \leq O_\epsilon(n^2)$ expected time resampling all the randomness each time we do this, it follows that this process takes $o(1)$ time in expectation. However, since our algorithm uses at most $(1 + 61\epsilon)\Delta$ colors in total as long as $\Delta(H) \leq 19\epsilon\Delta$, this ensures that we always maintain a $(1+61\epsilon)\Delta$-edge coloring of G, while only incurring an additive $o(1)$ factor in the expected worst-case update time. Thus, we have the following corollary.

Corollary C.5. *There exists a dynamic algorithm that, given a dynamic graph G that is initially empty and evolves by a sequence of κ edge insertions and deletions by means of an oblivious adversary, and a parameter $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ such that the maximum degree of G is at most*

Δ *at all times, maintains a* $(1+61\epsilon)\Delta$*-edge coloring of* G *and has an expected worst-case update time of* $O(\log^4(1/\epsilon)/\epsilon^9)$.

Finally, by amortizing over a sufficiently long sequence of updates and periodically resampling the random bits, we can get $O_\epsilon(1)$ amortized update time with high probability.

Corollary C.6. *There exists a dynamic algorithm that, given a dynamic graph* G *that is initially empty and evolves by a sequence of* κ *edge insertions and deletions by means of an oblivious adversary, and a parameter* $\Delta \geq (100\log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$ *such that the maximum degree of* G *is at most* Δ *at all times, maintains a* $(1+61\epsilon)\Delta$*-edge coloring of* G *and has an amortized update time of* $O(\log^4(1/\epsilon)/\epsilon^9)$ *over the whole update sequence w.h.p. as long as* κ *is a sufficiently large polynomial in* n.

Proof. We first make a further modification to the algorithm from Corollary C.5 so that, every n^2 updates, our algorithm samples new random bits for each edge in the same way as described above, taking $O_\epsilon(m)$ time in expectation. This splits up the update sequence $\sigma_1, \dots, \sigma_\kappa$ in *epochs* of length n^2. Let X_i be a random variable denoting the total update time of our algorithm during the i^{th} epoch. We have that $\mathbb{E}[X_i] = O(n^2\log^4(1/\epsilon)/\epsilon^9)$. Now note that, given the data structures that we use to implement our algorithm, there exists some polynomial $p(n)$ such that X_i is always at most $p(n)$. Let $X = \sum_i X_i$ be the total update time of our algorithm. Since we sample fresh randomness at the end of each epoch, the random variables $X_1, \dots, X_\tau$ are independent. Hence, we can apply Hoeffding bounds to get that $X = O(\kappa \log^4(1/\epsilon)/\epsilon^9)$ with probability at least $1 - \exp\left(-\tau \cdot \Theta_\epsilon(n^4)/p(n)^2\right)$. Hence, as long as $\tau = \Theta(\kappa/n^2)$ is a sufficiently large polynomial in n, $X = O(\kappa\log^4(1/\epsilon)/\epsilon^9)$ and the amortized update time of our algorithm is $O(\log^4(1/\epsilon)/\epsilon^9)$ with high probability. ☐

The rest of this section is now devoted to proving Lemmas C.2 and C.3. In Section C.2, we design the key data structure that we need to maintain a tentative coloring and associated failed edges. In Section C.3, we show how to use this data structure in order to achieve this and prove Lemma C.2. Finally, in Section C.4, we prove a (slight generalization of) Lemma C.3.

C.2 Key Data Structure

In this section, we design a dynamic data structure that will allow us to dynamically maintain a tentative coloring and the corresponding set of failed edges generated by this coloring. Given a dynamic graph $G = (V, E)$ and a parameter T, for each edge $e \in E$ our dynamic data structure maintains: the round of the edge, i_e, the sequence of colors of the edge, $c_e(1), \dots, c_e(K)$, and a *color index* $\ell_e \in \{0, \dots, K\}$. This defines a partition of the edges $S_1, \dots, S_{T+1}$ where $S_i = \{e \in E \mid i_e = i\}$, and an edge coloring $\tilde{\chi} : E \longrightarrow \mathcal{C} \cup \{\perp\}$ of G where $\tilde{\chi}(e) = c_e(\ell_e)$ and $c_e(0)$ is defined as $\perp$. The edge coloring $\tilde{\chi}$ then defines a set of failed edges $F = \{e \in E \mid \exists f \in N_{i_e}(e) \text{ such that } \tilde{\chi}(e) = \tilde{\chi}(f)\} \cup \{e \in E \mid \tilde{\chi}(e) = \perp\}$ and a collection of palettes $P_i(u) = [(1+\epsilon)\Delta] \setminus \tilde{\chi}(N_{<i}(u))$, $P_i(u,v) = P_i(u) \cap P_i(v)$. Our data structure maintains all of these objects and supports the following update and query operations.

Updates: The data structure can be updated with the following operations, as described below.

- Insert$(e, i, (c(1), \dots, c(K)))$: Inserts the edge e into the graph G and assigns the edge e round i, color sequence c, and color index 0.
- Delete(e): Deletes the edge e from the graph G.

- Set-Color-Index(e, ℓ): Sets the color index of the edge e to ℓ.
- Reset-Color(e): Sets the color index of the edge e to the smallest value ℓ such that $c_e(\ell) \in P_i(e)$, and to 0 if no such index exists. Returns the edge e if the color index of e changes, and null otherwise.

Queries: The data structure can answer the following types of queries, as described below.

- Node-Palette-Query(u, i, c): The input to this query is a node $u \in V$, an integer $i \in [T]$, and a color $c \in \mathcal{C}$. In response, the data structure outputs YES if $c \in P_i(u)$ and NO otherwise.
- Edge-Palette-Query(e, i, c): The input to this query is an edge $e \in E$, an integer $i \in [T]$, and a color $c \in \mathcal{C}$. In response, the data structure outputs YES if $c \in P_i(e)$ and NO otherwise.
- Failed-Edge-Query(e): The input to this query is an edge $e \in E$. In response, the data structure returns YES if $e \in F$ and NO otherwise.
- Color-Query(e): The input to this query is an edge $e \in E$. In response, the data structure returns the tentative color $\tilde{\chi}(e)$.

We now show how to implement this data structure so that each of these updates and queries run in $O_\epsilon(1)$ expected time. Our data structure also maintains other crucial internal data structures, which we describe below.

C.2.1 Implementation

We create hashmaps Round : $E \longrightarrow [T]$ and Color-Index : $E \longrightarrow [K]$ where Round$(e) = i_e$ and Color-Index$(e) = \ell_e$, allowing is to set and retrieve the rounds and color indices of edges in $O(1)$ (expected) time. The color sequences of edges are implemented as arrays, and we store a hashmap Color-Sequence that maps an edge e to the position of this array, allowing us to retrieve the color $c_e(\ell)$ for an edge $e \in E$ and $\ell \in [K]$ in $O(1)$ time. We implement the set F using a hashmap Failed, allowing us to insert, delete, and query the membership of an edge e in $O(1)$ time. Given some edge $e \in E$, we can then compute $\tilde{\chi}(e)$ in $O(1)$ time by retrieving ℓ_e and returning $c_e(\ell_e)$ if $\ell \neq 0$ and $\perp$ otherwise.

Internal data structures. Our data structure also maintains the following internal data structures that will be crucial for implementing our algorithm. We maintain hashmaps

$$\phi : V \times [T] \times \mathcal{C} \longrightarrow 2^{N_i(u)} \quad \text{and} \quad \Psi : V \times [T] \times \mathcal{C} \longrightarrow 2^{N_i(u)},$$

such that, for $u \in V$, $i \in [T]$, and $c \in \mathcal{C}$,

$$\phi_{u,i}(c) = \{e \in N_i(u) \mid \tilde{\chi}(e) = c\} \quad \text{and} \quad \Psi_{u,i}(c) = \{e \in N_i(u) \mid c \in c_e\}.$$

We take the convention that $\phi_{u,i}(\perp) = \varnothing$ and $\Psi_{u,i}(\perp) = \varnothing$. We also implement each set $\phi_{u,i}(c)$ as a hashmap, but maintain pointers between the elements of the set forming a doubly linked list. This allows for $O(1)$ time insertions, deletions, and membership queries, while also allowing us to return all the elements in the set in $O(|\phi_{u,i}(c)|)$ time. We implement the sets $\Psi_{u,i}(c)$ in the exact same way. By implementing ϕ as a hashmap, we avoid the preprocessing time of having to initialize each set in $\{\phi_{u,i}(c)\}_{u,i,c}$ upon creating the data structure and only initialize the set $\phi_{u,i}(c)$ when we want to insert an edge into $\phi_{u,i}(c)$, which takes $O(1)$ time. Otherwise, $\phi_{u,i}(c)$ points to null, and we know that the set is empty. Similarly, when $\phi_{u,i}(c)$ becomes empty, we can delete the map in $O(1)$ time, so that we only store maps that are non-empty.

Initialization. It follows that, upon initializing our data structure, we only need to create the maps that store the rounds, color sequences, color indices, and failed edges, as well as the 2 maps ϕ and Ψ. This takes $O(1)$ time in total.

C.2.2 Handling Updates

We now show how to maintain our data structure as we perform updates to the graph and the color indices.

Implementing Set-Color-Index**.** We first note that changing the color index ℓ_e of an edge e will *not* change the set $\Psi_{u,i}(c)$ or whether or not an edge $f \neq e$ is contained in the set $\phi_{u,i}(c)$ for any $u \in V$, $i \in [T]$, and $c \in \mathcal{C}$. Hence, in order to update the $\phi_{u,i}$ and $\Psi_{u,i}$ maps after a color index update for edge e, we only need to insert and remove e from the appropriate sets $\phi_{u,i}(c)$. On the other hand, some $O(1)$ many edges neighboring e might have to be added or removed from F after a color index update for e. Algorithm 6 shows how we implement the Set-Color-Index procedure in order to appropriately update all the data structures used by our algorithm.

Algorithm 6 Set-Color-Index(e, ℓ)

▷ Let u and v be the endpoints of e and $i = i_e$
$\ell_e^{\text{PREV}} \leftarrow \ell_e$ and $c^{\text{PREV}} \leftarrow \tilde{\chi}(e)$
$\ell_e \leftarrow \ell$ and $c \leftarrow \tilde{\chi}(e)$
▷ Update the ϕ maps
if $c^{\text{PREV}} \neq \perp$ **then**
  $\phi_{u,i}(c^{\text{PREV}}) \leftarrow \phi_{u,i}(c^{\text{PREV}}) \setminus \{e\}$
  $\phi_{v,i}(c^{\text{PREV}}) \leftarrow \phi_{v,i}(c^{\text{PREV}}) \setminus \{e\}$
if $c \neq \perp$ **then**
  $\phi_{u,i}(c) \leftarrow \phi_{u,i}(c) \cup \{e\}$
  $\phi_{v,i}(c) \leftarrow \phi_{v,i}(c) \cup \{e\}$
▷ Update the set of failed edges F
$X \leftarrow \{e\}$
if $|\phi_{u,i}(c)| = 2$ **then**
  $X \leftarrow X \cup \phi_{u,i}(c)$
if $|\phi_{v,i}(c)| = 2$ **then**
  $X \leftarrow X \cup \phi_{v,i}(c)$
if $|\phi_{u,i}(c^{\text{PREV}})| = 1$ **then**
  $X \leftarrow X \cup \phi_{u,i}(c^{\text{PREV}})$
if $|\phi_{v,i}(c^{\text{PREV}})| = 1$ **then**
  $X \leftarrow X \cup \phi_{v,i}(c^{\text{PREV}})$
for $f \in X$ **do**
  if $|\phi_{u,i}(\tilde{\chi}(f))| > 1$ or $|\phi_{v,i}(\tilde{\chi}(f))| > 1$ **then**
    $F \leftarrow F \cup \{f\}$
  else
    $F \leftarrow F \setminus \{f\}$

Lemma C.7. *The implementation of* Set-Color-Index *given by Algorithm 6 correctly updates the data structure and runs in time $O(1)$.*

Proof. Before the update, edge e is contained in both $\phi_{u,i}(c^{\text{PREV}})$ and $\phi_{v,i}(c^{\text{PREV}})$, and no other such sets. After the update, since $\tilde{\chi}(e)$ changes to c, we remove e from these sets and add it to $\phi_{u,i}(c)$ and $\phi_{v,i}(c)$, taking $O(1)$ time. After performing these operations, the ϕ maps are now correctly updated for the new tentative coloring. In order to see that our algorithm correctly updates the set F, note that the only edges that might need to be removed from F are those incident on u and v at round i that have tentative color c^{PREV}. However, if there is more than 1 edge incident on u (resp. v) at round i with tentative color c^{PREV} after updating the map ϕ, then no edge incident on u (resp. v) will be removed from F as all these edges will fail. Similarly, the only edges that might need to be added to F are those incident on u and v at round i that have tentative color c. However, if there are more than 2 edges incident on u (resp. v) at round i with tentative color c after updating the ϕ maps, then no edge incident on u (resp. v) will be added to F since these edges will have failed before the update. It follows that a total of $O(1)$ many edges might need to be added or removed from F and that we can identify them in $O(1)$ time. We can scan through all of these edges and for each one determine in $O(1)$ time whether it should or shouldn't be in F by checking if $|\phi_{u,i}(\tilde{\chi}(f))| > 1$ or $|\phi_{v,i}(\tilde{\chi}(f))| > 1$ which is true if and only f fails. □

Implementing Insert **and** Delete**.** We first note that inserting or deleting an edge $e = (u, v)$ that has tentative color $\perp$ will not cause any change in the map ϕ. Furthermore, it will not cause any edge $f \neq e$ to added or removed from F. Hence, we can insert or delete such edges and update the data structures easily. In order to delete an edge e that has $\tilde{\chi}(e) \neq \perp$, we can first call Set-Color-Index$(e, 0)$ and then assume we are deleting an edge with tentative color $\perp$. Algorithms 7 and 8 show how we can implement this. These algorithms clearly update all the data structures correctly and can be implemented to run in $O(K)$ time.

Algorithm 7 Insert$(e, i, (c(1), \dots, c(K)))$

$i_e \leftarrow i$
$c_e \leftarrow (c(1), \dots, c(K))$
$\ell_e \leftarrow 0$
$F \leftarrow F \cup \{e\}$
for $\ell' = 1 \dots K$ **do**
    $\Psi_{u,i}(c_e(\ell')) \leftarrow \Psi_{u,i}(c_e(\ell')) \cup \{e\}$
    $\Psi_{v,i}(c_e(\ell')) \leftarrow \Psi_{v,i}(c_e(\ell')) \cup \{e\}$

Algorithm 8 Delete(e)

Set-Color-Index$(e, 0)$
$i_e \leftarrow$ NULL
$c_e \leftarrow$ NULL
$\ell_e \leftarrow$ NULL
$F \leftarrow F \setminus \{e\}$
for $\ell' = 1 \dots K$ **do**
    $\Psi_{u,i}(c_e(\ell')) \leftarrow \Psi_{u,i}(c_e(\ell')) \setminus \{e\}$
    $\Psi_{v,i}(c_e(\ell')) \leftarrow \Psi_{v,i}(c_e(\ell')) \setminus \{e\}$

Implementing Reset-color**.** We implement the Reset-color update by scanning through the list of colors c_e and finding the smallest ℓ such that $c_e(\ell) \in P_{i_e}(e)$ by making calls to Edge-Palette-Query. Once we identify the smallest such ℓ, we call Set-Color-Index(e, ℓ) and return e if ℓ_e changes. If we cannot find such an index ℓ, we call Set-Color-Index$(e, 0)$ and return e if ℓ_e changes. As we will see in Section C.2.3, each edge palette query can be implemented to run in time $O(T)$. Since we make at most K such queries, one call to Set-Color-Index that takes $O(1)$ time, and the rest of the operations can be implemented in $O(1)$ time, it follows that Reset-color can be implemented to run in $O(KT)$ time. Algorithm 9 shows how we can implement this algorithm.

Algorithm 9 Reset-Color(e)

```
ℓ' ← ℓ_e
for ℓ = 1, ..., K do
    if Edge-Palette-Query(e, i_e, c_e(ℓ)) = YES then
        Set-Color-Index(e, ℓ)
        if ℓ' ≠ ℓ then
            return e
        return NULL
Set-Color-Index(e, 0)
if ℓ' ≠ 0 then
    return e
return NULL
```

C.2.3 Answering the Queries

Since we maintain the set F as a hashmap, given some edge e, we can answer Failed-Edge-Query on edge e in $O(1)$ time by directly checking whether the edge e is contained in F. Given some $u \in V$, $i \in [T]$, and $c \in \mathcal{C}$, we can check whether $c \in P_i(u) = [(1+\epsilon)\Delta] \setminus \tilde{\chi}(N_{<i}(u))$ by checking whether $\phi_{u,i'}(c) = \varnothing$ for all $i' < i$, which can be done in $O(i) \leq O(T)$ time. Hence, we can answer the query Node-Palette-Query in $O(T)$ time. Given some $e = (u, v) \in E$, $i \in [T]$, and $c \in \mathcal{C}$, we can check whether $c \in P_i(e)$ by making 2 calls to Node-Palette-Query and checking whether Node-Palette-Query $c \in P_i(u)$ and $c \in P_i(v)$. Hence, we can answer the query Edge-Palette-Query in $O(T)$ time. The fact that Color-Query can be implemented in $O(1)$ time follows immediately from the implementation.

C.3 Proof of Lemma C.2

Let $G = (V, E)$ be a dynamic graph that undergoes updates via a sequence of edge insertions and deletions, and let Δ be an upper bound on the maximum degree of G at any point in time. We now show how our data structure can be used to explicitly maintain the tentative coloring $\tilde{\chi}^{(t)}$ and set of failed edges $F^{(t)}$ defined by running Algorithm 4 on graph $G^{(t)}$ with parameters Δ and ϵ. Let i_e and c_e denote the round and color sequence sampled in advance for potential edge $e \in \binom{V}{2}$.

We first remark that, if we have that our data structure currently stores the graph $G^{(t-1)}$, each edge e receives the color sequence c_e and round i_e, and that the color index ℓ_e of each edge e equals the color index $\ell_e^{(t-1)}$ defined by Algorithm 4, then clearly the tentative coloring $\tilde{\chi}$ and the set of failed edges F maintained by our data structure equal $\tilde{\chi}^{(t-1)}$ and $F^{(t-1)}$. Hence, in order to be able to maintain $\tilde{\chi}^{(t-1)}$ and $F^{(t-1)}$, given the edge $e^\star$ that is either inserted or deleted during the t^{th} update, we need to appropriately update the graph maintained by our data structure (by inserting or deleting $e^\star$) and then update the color indices so that $\ell_e = \ell_e^{(t)}$ for all edges e. Given that our data structure is in the state described above, we now show how to do this efficiently.

Updating the data structure. When an edge $e^\star$ is inserted into the graph G, we run Algorithm 10 on input $e^\star$, which passes $e^\star$ to the data structure along with its round $i_{e^\star}$ and color sequence $c_{e^\star}$. When an edge $e^\star$ is deleted from the graph G, we run Algorithm 11 on input $e^\star$, which removes $e^\star$ from the data structure. In both cases, the algorithm then determines if $e^\star$ is dirty and then passes it to Algorithm 12 if this is the case. Algorithm 12 takes the set $A_{i_{e^\star}}^{(t)}$ and round $i_{e^\star}$ as inputs, where $i_{e^\star}$ is the first round containing dirty edges, and propagates the changes in the tentative coloring through the rounds. After the update is complete, we have that $\ell_e = \ell_e^{(t)}$ for all edges e in the graph.

Algorithm 10 INSERTION-UPDATE($e^\star$)

$\tilde{\chi}^{\text{PREV}}(e^\star) \leftarrow \perp$
INSERT($e^\star, i_{e^\star}, c_{e^\star}$)
$X \leftarrow \{\text{RESET-COLOR}(e^\star)\}$
PROPAGATE-CHANGES($X, i_{e^\star}$)

Algorithm 11 DELETION-UPDATE($e^\star$)

$\tilde{\chi}^{\text{PREV}}(e^\star) \leftarrow \tilde{\chi}(e^\star)$
DELETE($e^\star$)
$X \leftarrow \varnothing$
if $\tilde{\chi}^{\text{PREV}}(e^\star) \neq \perp$ **then**
    $X \leftarrow X \cup \{e^\star\}$
PROPAGATE-CHANGES($X, i_{e^\star}$)

Algorithm 12 PROPAGATE-CHANGES($X_{i'}, i'$)

for $i = i' + 1, \dots, T$ **do**
    $X_i' \leftarrow \bigcup_{e \in X_{i-1}} \Psi_{e,i}(\tilde{\chi}(e)) \cup \Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e))$
    $\tilde{\chi}^{\text{PREV}}(e) \leftarrow \tilde{\chi}(e)$ for all $e \in X_i'$
    $X_i \leftarrow X_{i-1}$
    for $e \in X_i'$ **do**
        $X_i \leftarrow X_i \cup \text{RESET-COLOR}(e)$

Here we let $\Psi_{e,i}(c)$ denote the set $\Psi_{u,i}(c) \cup \Psi_{v,i}(c)$, where $e = (u, v)$.

C.3.1 Correctness

Suppose that we call INSERTION-UPDATE($e^\star$) (resp. DELETION-UPDATE($e^\star$)) to update the data structure after the insertion (resp. deletion) of the edge $e^\star$ at time t. The following lemmas describe the behavior of our algorithm. We denote by ℓ_e and $P_i(u)$ the color indices and palettes maintained by our algorithm and by $\ell_e^{(t)}$ and $P_i^{(t)}(u)$ the color indices and palettes defined by Algorithm 4 on input $G^{(t)}$. Our objective is to show that $\ell_e = \ell_e^{(t)}$ for all edges e *after* the update is complete. Recall that we assume $\ell_e = \ell_e^{(t-1)}$ for all edges e *before* we the perform the update.

Lemma C.8. *Given any $X \subseteq S^{(t)}_{<i}$, we have that*

$$\Gamma^{(t)}_i(X) \cup \Lambda^{(t)}_i(X) \subseteq \bigcup_{e \in X} \Psi_{e,i}(\tilde{\chi}^{(t)}(e)) \cup \Psi_{e,i}(\tilde{\chi}^{(t-1)}(e)).$$

Proof. Let $e \in \Gamma^{(t)}_i(X)$. Then there exists some $f \in X$ such that $\tilde{\chi}^{(t)}(f) = \tilde{\chi}^{(t-1)}(e)$, so $\tilde{\chi}^{(t)}(f) \in c_e$ and hence $e \in \Psi_{f,i}(\tilde{\chi}^{(t)}(f))$. Now let $e \in \Lambda^{(t)}_i(X)$. Then there exists some $f \in X$ such that $\tilde{\chi}^{(t-1)}(f) = \tilde{\chi}^{(t)}(e)$, so $\tilde{\chi}^{(t-1)}(f) \in c_e$ and hence $e \in \Psi_{f,i}(\tilde{\chi}^{(t)}(f))$. □

Lemma C.9. *For all $i \in [T]$, we have that if $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{<i}$, then running* Recolor-Edge *on any $e \in S^{(t)}_i$ will set $\ell_e = \ell^{(t)}_e$.*

Proof. Since $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{<i}$, it follows that $\tilde{\chi}(e) = \tilde{\chi}^{(t)}(e)$ for all $e \in S^{(t)}_{<i}$, so $P_i(u) = P^{(t)}_i(u)$ for all $u \in V$. Given any edge $e \in S^{(t)}_i$, we then have that $P_i(e) = P^{(t)}_i(e)$, so when we run Recolor-Edge(e) we set the color index ℓ_e to the smallest ℓ such that $c_e(\ell) \in P^{(t)}_i(e)$ (or 0 if no such ℓ exists), which is the exact definition of $\ell^{(t)}_e$. □

Lemma C.10. *For all $i \geq i_e$, we have that $X_i = A^{(t)}_{\leq i}$. Furthermore, after our update procedure terminates, we have that $\ell_e = \ell^{(t)}_e$ for all $e \in E^{(t)}$.*

Proof. We prove this by induction. For all $i \geq i_{e^\star} + 1$, we show that the following are true at the start of the ith iteration of the **for** loop in Algorithm 12 (where we start from iteration $i_{e^\star} + 1$):

1. $X_{i-1} = A^{(t)}_{<i}$, and
2. $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{<i}$.

Base case. We begin by showing that these conditions all hold for $i = i_{e^\star} + 1$. Let $i^\star = i_{e^\star}$. We first show that $X_{i^\star} = A^{(t)}_{\leq i^\star}$. In the event that the t^{th} update is an insertion, we have that $X_{i^\star}$ is empty if $\tilde{\chi}^{(t)}(e^\star) = \perp$ (note that Reset-Color$(e^\star)$ returns $e^\star$ if and only if this is not the case), in which case the set $A^{(t)}_{\leq i^\star}$ is also empty, and $X_{i^\star} = \{e^\star\}$ if $\tilde{\chi}^{(t)}(e^\star) \neq \perp$, in which case $A^{(t)}_{\leq i^\star} = \{e^\star\}$. Similarly, in the event that the t^{th} update is an deletion, we have that $X_{i^\star}$ is empty if $\tilde{\chi}^{(t-1)}(e^\star) = \perp$, in which case $A^{(t)}_{\leq i^\star}$ is also empty, and $X_{i^\star} = \{e^\star\}$ if $\tilde{\chi}^{(t-1)}(e) \neq \perp$, in which case $A^{(t)}_{\leq i_e} = \{e^\star\}$. To see that $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{\leq i^\star}$ at the start of iteration $i^\star + 1$, note that $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{<i^\star}$ when we call Reset-Color$(e^\star)$ (since there are no dirty edges in the first $i^\star - 1$ rounds). Hence, by Lemma C.9, we have that $\ell_{e^\star} = \ell^{(t)}_{e^\star}$ at the start of iteration $i^\star + 1$. As this is the only edge in round $i^\star$ that can change its tentative color, the claim follows.

Inductive step. Suppose that the inductive hypothesis holds for $i_e + 1 \leq i \leq T$. We now show that it also holds for $i + 1$. By Lemma C.8 and Corollary B.9, we can see that

$$X'_i \supseteq \Gamma^{(t)}_i(X_{i-1}) \cup \Lambda^{(t)}_i(X_{i-1}) = \Gamma^{(t)}_i(A^{(t)}_{<i}) \cup \Lambda^{(t)}_i(A^{(t)}_{<i}) = A^{(t)}_i.$$

Since the algorithm scans through all of the edges in X'_i and calls Recolor-Edge on each of them, and we have that $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{<i}$, it follows by Lemma C.9 that we have $\ell_e = \ell^{(t)}_e$ for all $e \in S^{(t)}_{\leq i}$ at the end of the iteration (note that X'_i contains all of the edges at round i that change their color indices during this update). The algorithm places all of the edges $e \in X'_i$ that change their color index ℓ_e after calling Recolor-Edge(e) into X_i along with the edges in $A^{(t)}_{<i}$. Since $X'_i \subseteq A^{(t)}_i$, it follows that $X_i = A^{(t)}_{\leq i}$ at the end of the iteration. □

C.3.2 Update Time Analysis

Lemma C.11. *For all $i > i_{e^\star}$, $\mathbb{E}[|X_i'|] \le (32/\epsilon)\log(1/\epsilon) \cdot |X_{i-1}|$.*

Proof. For all $i > i_e$,

$$X_i' = \bigcup_{e \in X_{i-1}} \Psi_{e,i}(\tilde{\chi}(e)) \cup \Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e)),$$

which implies that

$$|X_i'| \le \sum_{e \in X_{i-1}} |\Psi_{e,i}(\tilde{\chi}(e))| + |\Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e))|.$$

Let $u \in V$, $i \in [T]$, and $c \in [(1+\epsilon)\Delta]$. Then for all $e \in N_i(u)$ we have that

$$\Pr[e \in \Psi_{u,i}(c)] = \Pr[c \in c_e] \le \frac{K}{(1+\epsilon)\Delta}$$

since c_e is a sequence of K colors sampled independently and uniformly at random from $[(1+\epsilon)\Delta]$. By linearity of expectation we get that $\mathbb{E}[|\Psi_{u,i}(c)|] \le |N_i(u)| \cdot K/((1+\epsilon)\Delta)$. Taking expectations on both sides and noting that $\mathbb{E}[|N_i(u)|] \le \epsilon(1-\epsilon)^{i-1}\Delta$ (see Lemma A.9), we get that

$$\mathbb{E}[|\Psi_{u,i}(c)|] \le \mathbb{E}[|N_i(u)|] \cdot \frac{K}{(1+\epsilon)\Delta} \le \epsilon K = \frac{8}{\epsilon}\log\frac{1}{\epsilon}.$$

It follows that

$$\mathbb{E}[|X_i'|] \le \sum_{e \in X_{i-1}} \mathbb{E}[|\Psi_{e,i}(\tilde{\chi}(e))|] + \mathbb{E}[|\Psi_{e,i}(\tilde{\chi}^{\text{PREV}}(e))|] \le |X_{i-1}| \cdot \frac{32}{\epsilon}\log\frac{1}{\epsilon}.$$

□

Lemma C.12. *Algorithms 10 and 11 run in $O(\log^4(1/\epsilon)/\epsilon^5 \cdot \mathbb{E}[|A^{(t)}|])$ expected time.*

Proof. Algorithms 10 and 11 run in $O(KT)$ time excluding the call to Propagate-Changes$(X, i_{e^\star})$, since the calls to Insert and Delete take $O(K)$ time, the call to Reset-Color$(e^\star)$ takes $O(KT)$ time, and the rest of the operations run in $O(1)$ time. The i^{th} iteration of Algorithm 12 (where we start from iteration $i_{e^\star}+1$) takes time $O(|X_{i-1}| + KT \cdot |X_i'|)$. Since we can return the sets $\Psi_{u,i}(c)$ in time proportional to their size, Line 2 runs in $O(|X_i'| + |X_{i-1}|)$ time. Lines 3-4 also run in time $O(|X_i'| + |X_{i-1}|)$. Lines 5-6 take $O(KT \cdot |X_i'|)$. It follows that the running time of these algorithms is

$$O(KT) + \sum_{i=i^\star+1}^{T} O(|X_{i-1}| + KT \cdot |X_i'|).$$

Taking expectations, applying Lemma C.11, and then taking expectations again, it follows that the expected running time of these algorithms is

$$O(KT) + \sum_{i=i^\star+1}^{T} O\left(\frac{1}{\epsilon^4}\log^3\frac{1}{\epsilon} \cdot \mathbb{E}[|X_{i-1}|]\right) = O\left(\frac{1}{\epsilon^4}\log^3\frac{1}{\epsilon}\right) \cdot \sum_{i=i^\star+1}^{T} \mathbb{E}[|X_{i-1}|].$$

Applying Lemma C.10, we can upper bound this by

$$O\left(\frac{1}{\epsilon^4}\log^3\frac{1}{\epsilon}\right) \cdot T \cdot \mathbb{E}[|A_{\le T}^{(t)}|] \le O\left(\frac{1}{\epsilon^5}\log^4\frac{1}{\epsilon} \cdot \mathbb{E}[|A^{(t)}|]\right).$$

□

C.4 Proof of Lemma C.3

Let $0 < \delta \leq 1$ be a constant. Then we have the following lemma.

Lemma C.13. *There exists a dynamic data structure that, given a dynamic graph $G = (V, E)$ that undergoes edge insertions and deletions, can explicitly maintain a $(2 + \delta)\Delta(G)$-edge coloring of G, has an expected update time of $O(1/\delta)$, and only changes the colors of $O(1)$ many edges per update.*

Proof. Our data structure maintains the graph G in a manner that allows edges to be inserted and deleted in $O(1)$ time (see Section C.2), and the coloring $\chi : E \longrightarrow [(2 + \delta)\Delta(G)]$ using a hashmap, allowing us to retrieve and set a color $\chi(e)$ in $O(1)$ time for any $e \in E$. Similarly, it maintains a hashmap $\psi : V \times [(2 + \delta)\Delta(G)] \longrightarrow E$ that maps nodes u and a color c to the edge e incident on u with $\chi(e) = c$, if such an edge exists. This algorithm maintains the invariant that each edge $e = (u, v)$ receives a color from $[(2 + \delta) \max\{\deg(u), \deg(v)\}]$.

Inserting an edge. When we insert an edge $e = (u, v)$ into the graph G, we sample a color c independently and u.a.r. from $[(2 + \delta) \max\{\deg(u), \deg(v)\}]$. Let $d^\star := \max\{\deg(u), \deg(v)\}$. The probability that this color is available at e, i.e. is contained in the palette $P(e) = P(u) \cap P(v)$ where $P(u) = [(2 + \delta)d^\star] \setminus \chi(N(u))$, is at least

$$\Pr[c \in P(e)] = \frac{|P(e)|}{(2+\delta)d^\star} = \frac{|P(u)| + |P(v)| - |P(u) \cup P(v)|}{(2+\delta)d^\star}$$

$$\geq \frac{(2+\delta)d^\star - \deg(u) + (2+\delta)d^\star - \deg(v) - (2+\delta)d^\star}{(2+\delta)d^\star} \geq \frac{\delta}{(2+\delta)} \geq \frac{\delta}{3}.$$

Using the hashmaps ψ_u and ψ_v we can check whether $c \in P(e)$ in $O(1)$ time. If $c \notin P(e)$, we sample another color in the same way, and repeat until we find a color $c \in P(e)$. Since these events that the sampled colors are in $P(e)$ are independent and all succeed with probability at least $\delta/3$, it follows that the expected number of colors that we have to sample before finding one in $P(e)$ is $3/\delta$. Hence, it takes us $O(1/\delta)$ time to find a color $c \in P(e)$ in expectation. Once we have found such a color, we set $\chi(e) \leftarrow c$, $\psi_u(c) \leftarrow e$, and $\psi_v(c) \leftarrow e$. Since the degrees of nodes cannot decrease during an edge insertion, the invariant is still satisfied.

Deleting an edge. When we delete an edge $e = (u, v)$ from the graph G, we first set $\psi_u(\chi(e)) \leftarrow$ NULL and $\psi_v(\chi(e)) \leftarrow$ NULL, and then set $\chi(e) \leftarrow$ NULL. However, the degrees of u and v decrease by one, so there might be edges that no longer satisfy the invariant. In particular, any edge f incident on u that no longer satisfies the invariant will have $\chi(f) \in [(2 + \delta)(\deg(u) + 1)] \setminus [(2 + \delta) \deg(u)]$. Since there are at most 3 such colors, we can use the hashmap ψ_u to identify the edges incident on u that receive one of these colors in $O(1)$ time. We then uncolor these edges, while ensuring to appropriately update the hashmaps, and recolor them using the same strategy as outlined above for an edge insertion. We do the same thing at node v. In total, we recolor at most 6 edges in G to ensure that we still satisfy the invariant, taking $O(1/\delta)$ time in expectation. □

D Implementing our Static Algorithm (Full Version)

We now show how to implement our static algorithm directly in order to obtain better performance than what follows immediately from our dynamic algorithm. We first show how to get linear time in expectation, and then expand on this to get linear time with high probability. The main result in this appendix is Theorem D.5, which is restated below.

Theorem D.1. *There exists an algorithm that, given a graph G with maximum degree Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, returns a $(1 + 61\epsilon)\Delta$-edge coloring of G in $O(m \log(1/\epsilon)/\epsilon^2)$ time with probability at least $1 - O(1/n^6)$.*

D.1 Linear Time in Expectation

It follows immediately from Corollary C.5 that our static algorithm can be implemented to run in $O(m \log^4(1/\epsilon)/\epsilon^9)$ expected time. We now show how to implement our static algorithm more directly using the data structure from Appendix C in order to get the following theorem.

Theorem D.2. *There exists an algorithm that, given a graph G with maximum degree Δ such that $\Delta \geq (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, returns a $(1+61\epsilon)\Delta$-edge coloring of G in $O(m \log(1/\epsilon)/\epsilon^2)$ expected time.*

Proof. We begin by using our data structure as a black box in order to show how our static algorithm can be implemented to run in $O(KTm)$ expected time. We then show how a white-box application can improve this to $O(Km)$ expected time.

Given the graph $G = (V, E)$, we begin by splitting the edges E into $\mathcal{E}_1, \dots, \mathcal{E}_\eta$ as defined in Algorithm 3. We do this by creating lists to store each of the $\mathcal{E}_j$ and scan through all of the edges in $e \in E$, assigning each to one of the $\mathcal{E}_j$ independently and u.a.r., taking $O(m)$ time. If we can now show how to implement Algorithm 4 so that it runs in expected time $g(m')$ on an input graph with m' edges, it follows that we can compute the failed edges and the tentative coloring in expected time $\sum_{j\in[\eta]} g(|\mathcal{E}_j|)$. By then noticing that Lemma C.13 immediately implies an expected linear time greedy algorithm, we get that the static algorithm runs in expected time $O(m) + \sum_{j\in[\eta]} g(|\mathcal{E}_j|)$. If the function g is linear and $g(x) = \Omega(x)$ (as it will be in the following cases), this then gives an expected running time of $O(g(m))$. This algorithm produces a $(1 + 61\epsilon)\Delta$-edge coloring with probability at least $1 - O(1/n^6)$. If it uses more than $(1 + 61\epsilon)\Delta$ many colors, we can afford to keep rerunning it using fresh randomness until we use at most $(1 + 61\epsilon)\Delta$ many colors, without increasing the asymptotic expected running time. (see Appendix C for a precise description of how to implement the resampling efficiently).

Black-box application of our data structure. Now we show how to implement Algorithm 4 so that it runs in $O(KTm)$ expected time given a graph $G = (V, E)$ with m edges. We begin by splitting the set E into $S_1, \dots, S_{T+1}$ by sampling a round i_e for each edge $e \in E$ and then placing e into a list containing the edges that are colored at round i_e, again taking $O(m)$ time in total. We then proceed to initialize our data structure, which takes $O(1)$ time. For $i = 1 \dots T$, we then insert all of the edges in S_i into the data structure (in any order) so that they are assigned a color index of 0 and hence left uncolored, and then call Reset-Color on each edge in S_i (again, in any order). By Lemma C.9, we have (by induction) that this correctly computes the color indices of all the edges in S_i as defined by the color sequences and rounds that we generate for the edges. Hence, we also compute the correct tentative colors and set of failed edges. Since Reset-Color runs in time $O(KT)$, and it takes $O(K)$ time to insert an edge into the data structure, it follows that the algorithm runs in time $O(KTm)$ since must do this for each edge exactly once.

White-box application of our data structure. In order to improve this to $O(Km)$, we notice that in this setting we can implement the query Edge-Palette-Query to run in in $O(1)$ expected time, improving the runtime of Reset-Color to $O(K)$, which gives the result. We do this by noticing that we can maintain a hashmap $\phi' : V \times \mathcal{C} \longrightarrow 2^{N(u)}$ where $\phi'_u(c) = \{e \in N(u) \mid \tilde{\chi}(e) = c\}$ and each $\phi'_u(c)$ is implemented in the same way as the $\phi_{u,i}(c)$. Since $\phi'_u(c) = \bigcup_{i=1}^T \phi_{u,i}(c)$, we can maintain the map ϕ' by appropriately updating the set $\phi'_u(c)$ every time we update one the sets

$\phi_{u,i}(c)$, incurring only $O(1)$ overhead. Since this $O(1)$ overhead does not change the asymptotic behavior of our data structure, this does not change the asymptotic running time of any of the updates or queries. When our algorithm now makes a call to Edge-Palette-Query after calling Reset-Color on an edge $e = (u, v)$ appearing in round i, we note that no edges in $S_{>i}$ have been inserted into the graph yet. Hence, we have that a color c is contained in $P_i(e)$ iff $|\phi'_u(c)| - |\phi_{u,i}(c)| = 0$ and $|\phi'_v(c)| - |\phi_{v,i}(c)| = 0$, which we can check in $O(1)$ time. □

D.2 Linear Time with High Probability

In order to obtain an algorithm that runs in $O_\epsilon(m)$ time with high probability, we need to obtain concentration. Our first obstacle is our use of hashmaps. We need to argue that we can implement these hashmaps so that not only can we handle insert, delete, and query operations in $O(1)$ expected time, but also so that these operations all take $O(1)$ time with high probability. For this, we use the following lemma which follows from [DadH90].

Lemma D.3 (Theorem 5.5, [DadH90])**.** *There exists a dynamic dictionary that, given a parameter k, can handle k insertion, deletion, and query operations, uses $O(k)$ space, and takes $O(1)$ worst-case time per operation with probability at least $1 - O(1/k^7)$.*

The second obstacle comes from our $O(\Delta)$ greedy algorithm, where we can make an unbounded number of queries to a hashmap. By applying the following concentration inequality for sums of geometric random variables given in [Bro11], we show that we only perform $O(m)$ many queries to the hashmap with high probability.

Lemma D.4. *Let $X_1, \dots, X_n$ be n independent geometric random variables with success probability p, and let $X = \sum_i X_i$. Then, for $0 < \epsilon \le 1$, we have that*

$$\Pr[X > (1+\epsilon)\mathbb{E}[X]] \le \exp\left(-\frac{\epsilon^2}{4}n\right).$$

By combining these tools, we can analyze our static algorithm more carefully, and get the following result.

Theorem D.5. *There exists an algorithm that, given a graph G with maximum degree Δ such that $\Delta \ge (100 \log n/\epsilon^4)^{(30/\epsilon)\log(1/\epsilon)}$, returns a $(1 + 61\epsilon)\Delta$-edge coloring of G in $O(m \log(1/\epsilon)/\epsilon^2)$ time with probability at least $1 - O(1/n^6)$.*

Proof. We begin by showing how we can use the dynamic dictionary from Lemma D.3 to implement the hashmaps in Algorithm 4 so that each insertion, deletion, and query made by the algorithm which we described in the preceding section takes $O(1)$ time with probability at least $1 - O(1/m^7)$, where m is the number of edges in the input graph $G = (V, E)$. This will immediately imply that we can implement Algorithm 4 to run in $O(Km)$ time with probability at least $1 - O(1/m^7)$. We first note that our data structure uses the maps ϕ, ϕ', Round, Color-Index, Color-Sequence, and Failed. Our implementation of Algorithm 4 does not need to use the map Ψ, so we can ignore all operations performed on this map in this context. As for the sets $\phi_{u,i}(c)$, note that they can only increase in size as we run the algorithm. Since the algorithm never accesses the elements in $\phi_{u,i}(c)$ after its size exceeds 2, we do not need to store the set once it becomes large enough, and instead we just store its size after this point. Hence, we do not need to implement them as hashmaps and can implement all operations on the set $\phi_{u,i}(c)$ in $O(1)$ worst-case time. The same applies to the sets $\phi'_u(c)$. Since we never need to access the elements in $\phi'_u(c)$, it is sufficient to simply keep track of its size, which can easily be achieved with a counter that can be updated in $O(1)$ time.

For each edge $e \in E$, we make one call to Insert(e) and one call to Reset-Color(e). It follows that, throughout the entire run of Algorithm 4, we perform $O(Km)$ many operations on the map ϕ' and $O(m)$ many operations on each of the other maps. Hence, we can implement each of these maps using the dynamic dictionary in Lemma D.3, initializing each one with a parameter of size $O(Km)$. It follows from a union bound that all the operations performed on all the hashmaps throughout the run of Algorithm 4 takes $O(1)$ time with probability at least $1 - O(1/(Km)^7) \geq 1 - O(1/n^7)$. By the arguments in the preceding section, it follows that the total time taken to subsample the graph and handle each call to Algorithm 4 for each of the subsampled graphs is $O(Km)$ with probability at least $1 - O(\eta/n^7) \geq 1 - O(1/n^6)$.

In order to implement our greedy algorithm so that it runs in $O(m)$ time with high probability, we again implement the hashmap ψ used by the algorithm using the dynamic dictionary in Lemma D.3, passing a sufficiently large parameter of order $O(m)$. However, it is not immediately clear that the algorithm performs at most $O(m)$ operations on the map ψ. We can observe that the insertion of the edge e into the greedy algorithm leads to $O(X_e)$ many operations on the map ψ, where X_e is a random variable denoting the number of colors sampled by the algorithm while trying to sample a color from the palette of edge e. Clearly, the value of X_e is $O(1)$ in expectation. It follows that, in expectation, we perform $\mathbb{E}[O(\sum_e X_e)] \leq O(m)$ many operations on the map ψ. In order to establish concentration, we can observe that $\{X_e\}_e$ is a collection of independent geometric random variables. Thus, we can apply the concentration inequality from Lemma D.4 to get that our algorithm performs at most $O(m)$ many operations on ψ with probability at least $1 - e^{-\Theta(m)}$. Conditioned on this event, all of the operations performed on our hashmap run in $O(1)$ worst-case time with probability at least $1 - O(1/m^7)$. Hence, the greedy algorithm runs in $O(m)$ time with probability at least $1 - O(1/m^7)$.

Putting everything together, we get that our static algorithm runs in $O(Km)$ time with probability at least $1 - O(1/n^6)$. Since the algorithm returns a $(1 + 61\epsilon)\Delta$-edge coloring with probability at least $1 - O(1/n^6)$, the lemma follows. □

E Concentration of Measure

In this appendix, we state the probabilistic tools that we use to establish concentration of measure throughout our paper. This appendix is essentially a subset of Appendix E in [BGW21].

E.1 Concentrartion Bounds

We now introduce some standard concentration bounds for independent random variables. The proofs of all of these bounds can be found in [DP09].

Proposition E.1 (Chrenoff Bounds). *Let X be the sum of n mutually independent indicator random variables $X_1, \dots, X_n$. Then, for any $\mu_L \leq \mathbb{E}[X] \leq \mu_H$, for all $\epsilon > 0$, we have that*

$$\Pr[X > (1 + \epsilon)\mu_H] \leq \exp\left(-\frac{\epsilon^2}{3}\mu_H\right),$$

$$\Pr[X < (1 - \epsilon)\mu_L] \leq \exp\left(-\frac{\epsilon^2}{2}\mu_L\right).$$

Proposition E.2 (Hoeffding Bounds). *Let X be the sum of n mutually independent indicator random variables $X_1, \dots, X_n$. Then, for all $t > 0$, we have that*

$$\Pr[X > \mathbb{E}[X] + t] \leq e^{-2t^2/n},$$

$$\Pr[X < \mathbb{E}[X] - t] \leq e^{-2t^2/n}.$$

Definition E.3 (Lipschitz Functions)**.** *Consider n sets $A_1, \ldots, A_n$ and a real valued function $f : A_1, \ldots, A_n \longrightarrow \mathbb{R}$. The function f satisfies the Lipschitz property with constants $d_1, \ldots, d_n$ if and only if $|f(x) - f(y)| \leq d_i$ whenever x and y differ only in the* ith *coordinate, for all $i \in [n]$.*

Proposition E.4 (Method of Bounded Differences)**.** *If f satisfies the Lipschitz property with constants $d_1, \ldots, d_n$, and $X_1, \ldots, X_n$ are independent random variables, then, for all $t > 0$, we have that*

$$\Pr[f < \mathbb{E}[f] + t] \leq \exp\left(\frac{2t^2}{d}\right),$$

$$\Pr[f > \mathbb{E}[f] - t] \leq \exp\left(\frac{2t^2}{d}\right),$$

where $d = \sum_i d_i^2$.

E.2 Negatively Associated Random Variables

We will sometimes need to get concentration around the sums for *negatively associated random variables*. We will need the following tools.

Definition E.5 (Negatively Associated Random Variables, [DP09, JDP83])**.** *We say that the random variables $X_1, \ldots, X_n$ are* negatively associated *(NA), if any two monotone increasing functions f and g defined on disjoint subsets of the variables in $\{X_i\}_i$ are negatively correlated. That is,*

$$\mathbb{E}[f \cdot g] \leq \mathbb{E}[f] \cdot \mathbb{E}[g].$$

Independent random variables are trivially NA.

Proposition E.6 (0-1 Principle, [DR96])**.** *Let $X_1, \ldots, X_n \in \{0, 1\}$ be binary random variables such that $\sum_i X_i \leq 1$. Then $X_1, \ldots, X_n$ are NA.*

Definition E.7 (Permutation Distribution)**.** *Let $x_1, \ldots, x_n$ be n values and let $X_1, \ldots, X_n$ be random variables taking on all permutations of $(x_1, \ldots, x_n)$ with equal probability. Then we call the collection of random varaibles $X_1, \ldots, X_n$ a permutation distribution.*

Proposition E.8 ([JDP83])**.** *Collections of random variables that form permutation distributions are NA.*

Proposition E.9 (NA Closure Properties, [DP09])**.**

- **Closure under products.** *If $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ are two independent families of random variables that are separately NA, then $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ is also NA.*
- **Disjoint Monotone Aggregation.** *If $X_1, \ldots, X_n$ are NA, and $f_1, \ldots, f_k$ are monotone (either all increasing or all decreasing) functions defined on disjoint subsets of the random variables, then $f_1(X_1, \ldots, X_n), \ldots, f_k(X_1, \ldots, X_n)$ are NA.*

The Chernoff-Hoeffding bounds from Propositions E.1 and E.2 extend to the case where the random variables are NA [DP09].