

FASTER DYNAMIC $(\Delta + 1)$ -COLORING AGAINST ADAPTIVE ADVERSARIES

MAXIME FLIN AND MAGNÚS M. HALLDÓRSSON

ABSTRACT. We consider the problem of maintaining a proper $(\Delta + 1)$ -vertex coloring in a graph on n -vertices and maximum degree Δ undergoing edge insertions and deletions. We give a randomized algorithm with amortized update time $\tilde{O}(n^{2/3})$ against adaptive adversaries, meaning that updates may depend on past decisions by the algorithm. This improves on the very recent $\tilde{O}(n^{8/9})$ -update-time algorithm by Behnezhad, Rajaraman, and Wasim (SODA 2025) and matches a natural barrier for dynamic $(\Delta + 1)$ -coloring algorithms. The main improvements are in the densest regions of the graph, where we use structural hints from the study of distributed graph algorithms.

1. INTRODUCTION

In the $(\Delta + 1)$ -coloring problem, we are given a graph with maximum degree Δ and must assign to each vertex v a color $\varphi(v) \in \{1, 2, \dots, \Delta + 1\}$ such that adjacent vertices receive different colors. Despite its simplicity in the classical RAM model, this problem is surprisingly challenging in more constrained models such as distributed graphs [Lin92, BEPS16, HSS18, CLP20], sublinear models [ACK19, CFG⁺19, CDP21, AY25], dynamic graphs [BCHN18, HP22, BGK⁺22, BRW25] and even recently communication complexity [FM24, CMNS24].

This paper presents a *fully dynamic* algorithm for the $(\Delta + 1)$ -coloring problem. The vertex set is fixed of size n , while edges can be inserted or deleted as long as the maximum degree remains at most Δ at all times. We assume Δ is known in advance. As the graph changes, our algorithm must maintain a $(\Delta + 1)$ -coloring. The time the algorithm takes to process an edge insertion or deletion is called the *update time* and our aim is to minimize it.

The study of dynamic algorithms for $(\Delta + 1)$ -coloring was initiated by [BCHN18], which provided a randomized algorithm with $O(\log n)$ amortized update time. Independently, the authors of [BGK⁺22] and [HP22] provided randomized algorithms with $O(1)$ amortized update time. The caveat with those algorithms is that they assume updates are *oblivious*: all edge insertions and deletions are fixed in advance and may not adapt to the colors chosen by the algorithm.

Maintaining a coloring against an adaptive adversary — that may base the updates on past decisions by the algorithm — is significantly harder, as the adversary can force the algorithm to recolor vertices at every update by connecting same-colored vertices. Until very recently, no $o(n)$ update time algorithm was known for $(\Delta + 1)$ -coloring against adaptive adversaries. In a recent breakthrough, Behnezhad, Rajaraman, and Wasim [BRW25] presented a randomized algorithm with $\tilde{O}(n^{8/9})$ amortized update time against adaptive adversaries¹. In this paper, we significantly improve the runtime, obtaining the following result.

Theorem 1. *There exists a randomized fully dynamic algorithm maintaining a $(\Delta + 1)$ -coloring against adaptive adversaries in $\tilde{O}(n^{2/3})$ amortized update time with high probability.*

This work was supported by the Icelandic Research Fund grants 217965 and 2310015.

¹In this paper, we use $\tilde{O}(f(n)) := O(f(n) \cdot \text{poly}(\log n))$ to hide polylogarithmic factors.

At its core, our algorithm is based on a structural decomposition of the vertices into four layers so that vertices of the bottom layer can be colored efficiently by maintaining and searching small sets. Vertices of higher layers are colored by random color trials enhanced with a *color stealing* mechanism. Specifically, when a vertex cannot use a color due to a lower-level neighbor, our algorithm steals the color and recolors the lower-level vertex instead. Interestingly, the layers correspond to the order in which streaming and distributed algorithms build their colorings.

In addition to the runtime improvement, our algorithm approaches a natural barrier for dynamic coloring algorithms against adaptive adversaries. The algorithm of [BRW25] was primarily limited by the challenge of efficiently recoloring dense regions of the graph. In contrast, our algorithm shifts this bottleneck to the recoloring of sparser vertices and the handling of random color trials in general. We elaborate on this in the following discussion.

1.1. Discussion & Open Problems. Before we explain why $n^{2/3}$ is a natural barrier, we first establish why $n^{1/2}$ poses a fundamental limit for dynamic algorithms based on random color trials. Observe that this limitation extends to simpler problems, such as 2Δ -coloring.

The bottleneck for recoloring a vertex lies in efficiently verifying whether a color is available. One possibility is to iterate through all neighbors in $O(\Delta)$ time; another is to iterate through color classes (i.e., the set of vertices with a given color). At best, all color classes contain $O(n/\Delta)$ vertices each. So, any algorithm that checks arbitrary or random colors using the former method when Δ is small and the latter when Δ is large therefore has $O(n^{1/2})$ update time at best. Recall that an adaptive adversary can trigger recoloring at every update, preventing any opportunity for amortization.

To reduce the number of colors to $\Delta + 1$, the approach of [EPS15, HSS18, ACK19, CLP20, BRW25] and ours is to compute a partial coloring in which (sparse) vertices have many pairs of neighbors colored the same. The randomized argument underlying this approach is fragile, requiring a complete rerun every $\Theta(\Delta)$ update. Since it is based on random color trials at every vertex, it runs in $O(n^2/\Delta)$ time. Amortized and balanced with the $O(\Delta)$ update time for small Δ , we obtain the $n^{2/3}$ update time.

Problem 1. *Does there exist a fully dynamic $(\Delta + 1)$ -coloring with $o(n^{2/3})$ update time that remains robust against adaptive adversaries?*

1.2. Organization of the Paper. In Section 2, we provide a high-level overview of our algorithm and compare it with [BRW25]. Section 3 introduces the sparser-denser decomposition that forms the foundation of our approach. The core of our dynamic algorithm, along with the proof of Theorem 1, is presented in Section 4. Finally, we defer the least novel part of our algorithm – the periodic recomputation of a fresh coloring of the whole graph – to Section 5.

2. OVERVIEW

We present the key concepts underlying Theorem 1, deliberately simplifying certain technical details for the sake of intuition. We focus on the case where $\Delta \geq n^{2/3}$, as an update time of $O(\Delta)$ is straightforward to achieve by examining the adjacency list of the vertices.

The Sparser-Denser Decomposition. We employ a modified version of the structural decomposition introduced by Reed [Ree98], and further explored in [HSS18, ACK19], which partitions the vertices into sparse vertices and dense clusters. We classify vertices into clusters based on their average density. Vertices in clusters with low average density are designated as *sparser*, while those in clusters with higher average density are considered *denser* (see Definition 3.3). This method is based on structural observations about the mean local density within a cluster

(see Lemma 3.6), and provides a significant efficiency gain compared to the specific form of decomposition used in [BRW25].

Coloring the Sparser Vertices. The strategy for coloring sparser vertices is essentially the same as in [BRW25]. Updates are performed in *phases* of length $t = \Theta(n^{2/3})$, with each phase beginning by computing a *fresh coloring* of the graph. This ensures that the adversary cannot decrease the number of available colors below t because each sparser vertex starts with $2t$ available colors (see Proposition 4.1) and the algorithm recolors at most one vertex per update. As the adversary updates the graph, we use the observation from [ACK19, BRW25] that a vertex with t available colors can be recolored in $\tilde{O}(n^{2/3})$ time by random color trials. Note that the cost of computing a fresh coloring, amortized over updates of the following phase, is $\tilde{O}(n^{2/3})$.

Coloring the Denser Vertices. We treat two types of vertices differently in each dense cluster. We exploit the fact that most vertices have sparsity not much greater than the average sparsity of their cluster. These vertices — the *inliers* — have $O(n^{2/3})$ neighbors outside their cluster and $O(n^{2/3})$ anti-neighbors within their cluster. This allows for a compact representation of the colors available to inliers, and a simple deterministic search suffices to recolor them. The remaining vertices — the *outliers* — are recolored using random color trials, similar to the strategy for sparser vertices. The key idea is that outliers may *steal* a color from an inlier neighbor, causing the inlier to be recolored as described above. Since most colors are available for stealing, only a few random trials are required. (See Lemma 4.5 for details.)

Maintaining the Color Balance. To guarantee efficient color trials, we ensure that each color class contains $\tilde{O}(n^{1/3})$ vertices. This condition holds for the sparser vertices by an adaptation of [BRW25] (see Lemma 4.9). For the denser vertices, we maintain a *Dense Balance Invariant*: each cluster contains at most two vertices of each color. To ensure that enough colors are available for inliers, we maintain a second invariant.

The *Matching Invariant* ensures that enough colors are repeated within each cluster. These repeated colors form a *colorful matching* [ACK19], which can be maintained efficiently (again) using random color trials. With the Matching Invariant in place, we can ensure that inliers receive a color that is unique within their cluster, thereby preserving the Dense Balance Invariant.

It is noteworthy how extensively our algorithm employs *color stealing*: sparser vertices can steal colors from matched vertices, which can steal from outliers, and outliers can steal from inliers.

Comparison with [BRW25]. The parameter used in [BRW25] to distinguish between sparser and denser vertices is the ε of the structural decomposition [Ree98, HSS18], ultimately set to $\varepsilon = \frac{\Delta^{1/5}}{n^{2/5}}$. While this choice allows for a uniform treatment of dense vertices, it is costly due to its quadratic dependence on (local) sparsity. Specifically, sparse vertices have $\Omega(\varepsilon^2 \Delta)$ available colors, which limits the length of each phase and increases the amortized cost of recomputing fresh colorings. Additionally, maintaining the decomposition itself incurs an update time of $\tilde{O}(\varepsilon^{-4})$. In contrast, we keep ε *constant* and instead use the average sparsity of dense clusters directly as a parameter to classify vertices as sparser or denser. Our method employs a two-pronged approach to recoloring denser vertices (inliers vs. outliers), with both steps relatively simple. While [BRW25] reasons about dense vertices through perfect matchings in the palette graph of [ACK19], we work directly with vertices and colors. The augmenting paths in their palette graph are similar to our color-stealing mechanism.

A notable technical difference concerns the colorful matching: while [BRW25] maintains a *maximal* matching, we find that a sufficiently large matching suffices, allowing for simpler updates.

Recent years have seen a number of exciting results on $(\Delta + 1)$ -coloring across models, all made possible by the power of random color trials and, in most cases, with the help of Reed’s structural decomposition [Ree98]. We find particularly interesting that analyzing the *average sparsity/density* of its clusters (inspired by [HNT21, FGH⁺23]) enabled us to improve on [BRW25] down to a natural barrier. In particular, the inliers/outliers dichotomy for balancing the cost of coloring dense and sparse vertices may be of independent interest.

2.1. Notation. For an integer $k \geq 1$, we write $[k] = \{1, 2, \dots, k\}$. Throughout the paper, the input graph has a fixed vertex set $V = [n]$. The neighborhood of v is $N(v)$ and we write $\deg(v) = |N(v)|$ its degree. For a set $S \subseteq V$, let $N_S(v) = N(v) \cap S$ be the set of neighbors in the set. We assume that Δ is known in advance and that, at all times, every vertex has degree at most Δ . A non-adjacent pair of nodes u, v is said to be an *anti-edge*.

A *partial coloring* is a function $\varphi : V \rightarrow [\Delta + 1] \cup \{\perp\}$, where $\varphi(v) = \perp$ indicates that v is not colored. For a set S of nodes, let $\varphi(S) = \{\varphi(v) : v \in S\}$. A coloring is *total* when all vertices are colored, i.e., when $\perp \notin \varphi(V)$. The coloring is *proper* if, for every edge $\{u, v\}$, either $\varphi(u) \neq \varphi(v)$ or $\perp \in \{\varphi(u), \varphi(v)\}$. In this paper, all colorings are implicitly proper. The *palette* of v is $L(v) = [\Delta + 1] \setminus \varphi(N(v))$, i.e., the set of colors not used by neighbors of v . The colors of the palette are also said to be *available* (for v).

We say that an event holds “with high probability” — often abridged to “w.h.p.” — when it holds with probability at least $1 - n^{-c}$ for a sufficiently large constant $c > 0$.

The algorithm frequently uses classic sets and operations that can be implemented efficiently using, say, a red-black tree. They support insertions, deletions, and uniform sampling in $O(\log n)$ worst-case update time.

3. THE SPARSER-DENSER DECOMPOSITION

Our algorithm uses a modified version of the sparse-dense decomposition introduced by [Ree98] and further extended in [HSS18, ACK19] for $(\Delta + 1)$ -coloring in distributed and streaming models. For some small $\varepsilon \in (0, 1)$, these decompositions partition the vertices into a set of $\Omega(\varepsilon^2 \Delta)$ -sparse vertices and a number of dense clusters called ε -almost-cliques. We introduce now the key definitions of sparsity and almost-cliques.

Definition 3.1. Vertex v is ζ -sparse if $G[N(v)]$ contains at most $\binom{\Delta}{2} - \zeta \Delta$ edges.

Definition 3.2. A set $D \subseteq V$ of vertices is an ε -almost-clique if

- (1) $|D| \leq (1 + \varepsilon)\Delta$, and
- (2) every $v \in D$ has $|N(v) \cap D| \geq (1 - \varepsilon)\Delta$.

For a vertex v in an almost-clique D , the set of its neighbors outside D and the set of its non-neighbors inside D play a key role in our algorithm. We denote $E(v) = N(v) \setminus D$ and $A(v) = D \setminus (N(v) \cup \{v\})$ the sets of *external*- and *anti-neighbors*, respectively. We use $e_v = |E(v)|$ and $a_v = |A(v)|$ to denote their size.

As explained earlier, we differentiate between sparse and dense vertices differently from [Ree98, HSS18, ACK19, BRW25]. Namely, it suffices for us that an almost-cliques is dense *on average*. To make this precise, let $e_D = \sum_{v \in D} e_v / |D|$ and $a_D = \sum_{v \in D} a_v / |D|$ be the average external- and anti-degree in D , respectively. We can now state the definition of our sparser-denser decomposition.

Definition 3.3. Let $\varepsilon \in (0, 1)$, $\zeta \in [1, \Delta]$ and G a graph with maximum degree Δ . An (ε, ζ) -sparser-denser decomposition of G is a partition of its vertices into sets S, D_1, \dots, D_r such that

- (1) every $v \in S$ is ζ -sparse, and
- (2) every $D = D_i$ is an ε -almost-clique such that $a_D + e_D \leq O(\zeta/\varepsilon^2)$.

To compare our decomposition to that of [Ree98, HSS18, ACK19], we use their core decomposition with a constant ε but choose a sparsity parameter ζ and let S be the $\Omega(\varepsilon^2\Delta)$ -sparse vertices along with the almost-cliques for which $a_D + e_D \gg_\varepsilon \zeta$. This results only in ζ -sparse vertices due to a structural results on the sparsity of dense vertices (see Lemma 3.6). Henceforth, we refer to vertices of S as to the *sparser* vertices and vertices of D as the *denser* vertices.

As explained in Section 2, we wish to color most of the denser vertices by iterating over their external- and anti-neighbors. The vertices for which this is feasible are called *inliers*:

Definition 3.4. For each $D = D_i$ in an (ε, ζ) -sparser-denser decomposition, define

$$I_D = \{v \in D : e_v \leq 8e_D \text{ and } a_v \leq 8a_D\} \quad \text{and} \quad O_D = D \setminus I_D.$$

where vertices of I_D (resp. O_D) are called *inliers* (resp. *outliers*).

Note that by Markov's inequality, at least three quarters of each D_i are inliers.

3.1. Maintaining the Decomposition. In this section, we explain how the fully dynamic algorithm of [BRW25] for maintaining Reed's decomposition against an adaptive adversary can be adapted to our needs:

Lemma 3.5. There exists a universal constant $\delta \in (0, 1)$ for which the following holds. Let $\varepsilon \in (0, 1/6)$ and $1 \leq \zeta \leq \varepsilon^2 \cdot \delta\Delta$. There exists a randomized fully dynamic algorithm that maintains array $\text{part}[1 \dots n]$ that induces a partition of the vertices into sets $S = \{v : \text{part}[v] = \perp\}$ and $D_i = \{v : \text{part}[v] = i\}$ such that, w.h.p.,

- (S, D_1, \dots, D_r) is an (ε, ζ) -refined partition,
- for each $D_i \neq \emptyset$, it maintains the set F_{D_i} of anti-edges in $G[D_i]$, and
- for each $v \in D_i$, it maintains the sets $A(v)$ and $E(v)$.

The amortized update time against an adaptive adversary is $O(\varepsilon^{-4} \log n)$ with high probability.

Our algorithm builds on an ε -almost-clique decomposition, as defined by [ACK19]: a vertex partition V_{sp}, C_1, C_2, \dots where the nodes in V_{sp} have $\Omega(\varepsilon^2\Delta)$ -sparsity and the C_i 's are $\Theta(\varepsilon)$ -almost-cliques. We filter out the C_i 's with sparsity more than ζ and add them into V_{sp} to obtain the set S . The key structural result for our partition is the following lemma:

Lemma 3.6. Let $\varepsilon \in (0, 1/75)$ and $\delta \in (0, 1)$. Suppose $V_{sp}, C_1, C_2, \dots, C_r$ is a vertex partition such that every vertex in V_{sp} is $(\varepsilon^2 \cdot \delta\Delta)$ -sparse and each C_i is an ε -almost-clique. Then, every $v \in C_i$ such that $e_{C_i} + a_{C_i} \geq 16/(\delta\varepsilon^2)$ is $\frac{\delta\varepsilon^2}{50}(a_{C_i} + e_{C_i})$ -sparse.

Proof. **TOPROVE 0** ■

To maintain our sparse-denser decomposition, we use the algorithm of [BRW25] for the almost-clique decomposition.

Proposition 3.7 ([BRW25, Theorem 4.1]). For every $n \gg 1$ and $\eta \in (0, 3/50)$, there exists a fully dynamic randomized algorithm that maintains a vertex partition V_{sp}, C_1, \dots, C_r of any n -vertex graph with maximum degree Δ such that, w.h.p.,

- (1) vertices of V_{sp} are $\Omega(\eta^2\Delta)$ -sparse,

- (2) each C_i is a (10η) -almost-clique,
- (3) for every vertex it maintains the sets $N(v) \cap V_{sp}$ and $N(v) \cap (C_1 \cup \dots \cup C_r)$, and
- (4) for every almost-clique C_i , it maintains the set F_i of anti-edges in C_i .

The amortized update time against an adaptive adversary is $O(\eta^{-4} \log n)$ with high probability.

We can now prove that a sparser-denser decomposition can indeed be maintained.

Proof. **TOPROVE 1** ■

3.2. Clique Palette, Colorful Matching & Accounting Lemma. Similarly to [ACK19, FGH⁺24], we wish to assign dense vertices colors that are unused by vertices of their almost-clique. Together these form the *clique palette*.

Definition 3.8. For any (possibly partial) coloring φ of G , and almost-clique D , the clique palette $L(D)$ is the set of colors unused by vertices of D . More formally, $L(D) = [\Delta + 1] \setminus \varphi(D)$.

To ensure the clique palette always contains colors, we give the same color to a pair of vertices in the almost-clique. We say a color is *repeated* or *redundant* in $D \subseteq V$ when at least two vertices of D hold it. Assadi, Chen, and Khanna [ACK19] introduced the notion of a *colorful matching*: if M colors are repeated in D , there exists an M -sized anti-matching in $G[D]$ where the endpoints of a matched anti-edge have the same color.

Each redundant color in D gives its vertices some slack, while using the clique palette forbids at most one color for each anti-neighbor. Accounting for both, we obtain the following bound on the number of colors in the clique palette available for each vertex:

Lemma 3.9 (Accounting Lemma). *Let φ be a (possibly partial) coloring and D an almost-clique with a colorful matching of size M . For every $v \in D$, the number of colors available for v in the clique palette is at least*

$$|L(D) \cap L(v)| \geq \Delta - \deg(v) + M - a_v + [\# \text{uncolored vertices in } D \cup E(v)] .$$

Proof. **TOPROVE 2** ■

4. THE DYNAMIC ALGORITHM

In this section, we prove **Theorem 1**. We begin with the definition of phases and describe the properties of the coloring at the beginning of each phase. We then explain how the algorithm maintains a proper coloring over one phase. The analysis has three parts: first, we show that the algorithm maintains some invariants throughout the phase (**Section 4.1**); second, we prove that recoloring denser vertices is efficient (**Section 4.2**); finally, we analyze the re-coloring of sparser vertices (**Section 4.3**). Together, they imply **Theorem 1** (**Section 4.4**).

When $\Delta \leq O(n^{2/3})$, the naive deterministic algorithm that scans the entire neighborhood of vertices before recoloring them (when necessary) has worst-case update time $O(\Delta) \leq O(n^{2/3})$. We henceforth assume that $\Delta \geq \Omega(\zeta/\varepsilon^2) = \Omega(n^{2/3})$, for a sufficiently large constant where $\varepsilon := 1/110$ and $\zeta := n^{2/3}$ are the parameters of the sparser-denser decomposition.

Phases & Fresh Coloring. We divide updates into *phases* of t insertions and deletions and analyse the algorithm phase by phase. At the beginning of each phase, we compute a fresh coloring of the graph.

A phase is $t := \gamma \cdot \zeta$ updates long, where γ is the constant from **Proposition 4.1**.

The algorithm recolors at most one vertex in S per update, so the adversary can only remove one color per update from $L_S(v) := [\Delta + 1] \setminus \varphi[N_S(v)]$. By recomputing a fresh coloring every t updates, we ensure that sparser vertices always have $t = \gamma \cdot \zeta$ colors available, as they start the phase with more than $3t$ available colors.

Proposition 4.1. *There exists a universal constant $\gamma \in (0, 1)$ such that the following holds. For $\zeta \geq \Omega(\log n)$ for some large hidden constant, let S, D_1, \dots, D_r be a (ε, ζ) -decomposition of an n -vertex graph with maximum degree Δ . W.h.p., **RefreshColoring** runs in $\tilde{O}(n + n^2/\zeta)$ time, producing a total coloring φ such that*

- (Sparse Slack) every sparse vertex v has $|L_S(v)| \geq 3\gamma \cdot \zeta$,
- (Sparse Balance) $|\Phi[\chi] \cap S| = O(n/\zeta + \log n)$ for every $\chi \in [\Delta + 1]$,
- (Dense Balance) in each D_i , every color is used at most twice, and
- (Matching) each D_i has at least $\lfloor 8a_{D_i} \rfloor$ redundant colors.

The description of **RefreshColoring** and the proof of **Proposition 4.1** are deferred to **Section 5** to preserve the flow of the paper. Henceforth, we say that “**RefreshColoring** succeeded” if it ends and the coloring it produces has all the aforementioned properties. Throughout each phase, we maintain the Dense Balance and Matching properties given by **RefreshColoring** as invariants — and refer to them as the *Dense Balance Invariant* and *Matching Invariant*.

During a phase, it helps to keep the sparser-denser partition fixed. A simple adaptation of the algorithm from **Lemma 3.5** allows us to make this assumption. We emphasize that during a phase, the adversary may increase or decrease external- and anti-degrees of vertices and their averages. However, given the phase length, it does not materially affect the decomposition.

Lemma 4.2. *Let δ, ε and ζ be as in **Lemma 3.5**. There is an algorithm with $O(\varepsilon^{-4} \log n)$ amortized update time that maintains a vertex partition such that*

- at the beginning each phase, the partition is an (ε, ζ) -decomposition,
- the vertex partition does not change during phases,
- at all times, denser vertices have $e_v \leq 2\varepsilon\Delta$ and $a_v \leq 3\varepsilon\Delta$, and
- for each $D = D_i$, the values of a_D and e_D increase or decrease by at most one within a phase.

Proof. **TOPROVE 3** ■

The Dynamic Algorithm. Updates are triggered by calls to **Insert** and **Delete**. They maintain the sparser-denser partition as described in **Lemma 4.2** and, when a new phase begins, recompute a coloring from scratch using **RefreshColoring**. During phases, **Insert** and **Delete** update the data-structures and respectively call **RecolorInsert** and **RecolorDelete** to handle color conflicts.

Upon insertion of $\{u, v\}$, **RecolorInsert** verifies if it creates a color conflict (line 4), in which case it uncolors u . In most cases, it suffices to recolor u using **RecolorSparse** or **RecolorDense** depending on whether $v \in S$ or not (lines 10 to 13). The one exception to this concerns *matched vertices*. A denser vertex is *matched* if and only if its color is repeated by another vertex of its almost-clique. Maintaining sufficiently many matched vertices in each almost-clique (see Matching Invariant) is necessary for **RecolorDense**. If, after an update, there are not enough matched vertices — either because the almost-clique lost an anti-edge (line 6) or because a_D increased (line 2) — we call **IncrMatching**(D), which creates a new redundant color in D . To same-color pairs of (non-adjacent) vertices in D , we use **RecolorMatching**. It proceeds similarly in recoloring sparse vertices and outliers: random color trials in $[\Delta + 1]$ and possibly stealing a color from an inlier in D .

Algorithm 1: Updates handlers during a phase, when $\Delta \geq \Omega(n^{2/3})$

```

1 function RecolorDelete( $u, v$ ):
2   if  $part[u] = part[v] = i$  and  $|M_{D_i}| < \lfloor 8a_{D_i} \rfloor$  then IncrMatching( $D_i$ )
3 function RecolorInsert( $u, v$ ):
4   if  $\varphi[u] = \varphi[v]$  then
5     Color( $u, \perp$ )                                // uncolor  $u$ 
6     if  $matched[u] = w \neq \perp$  then
7        $matched[u], matched[w] \leftarrow \perp$           // unmatch vertices
8        $M_D \leftarrow M_D - \varphi[w]$                 // remove the color from the matching
9   if  $part[u] = part[v] = i$  and  $|M_{D_i}| < \lfloor 8a_{D_i} \rfloor$  then IncrMatching( $D_i$ )
10  if  $\varphi[u] = \perp$  then
11    if  $u \in S$  then RecolorSparse( $u$ )
12    else if  $matched[u] = w \neq \perp$  then RecolorMatching( $u, w$ )
13    else RecolorDense( $u$ )

```

Let us list the data-structures we use. The coloring is stored as an array $\varphi[1 \dots n]$ such that $\varphi[v]$ is the color of v . We also maintain the color class $\Phi[\chi] = \{v \in V : \varphi[v] = \chi\}$ for each $\chi \in [\Delta + 1]$. When we change the color of a vertex, we use the function $\text{Color}(v, \chi)$ that sets $\varphi[v]$ to χ and updates color classes accordingly. To keep track of redundant colors, we maintain:

- an array $matched[1 \dots n]$ such that $matched[u] = v \neq \perp$ if u and v are in the same almost-clique and colored the same, otherwise $matched[u] = \perp$;
- for each almost-clique D_i , the set M_{D_i} of repeated colors in D_i .

We now explain how the recoloring procedures used in [Algorithm 1](#) work for sparser, denser and matched vertices.

Sparser Vertices. For recoloring a vertex in S , we sample random colors in $[\Delta + 1]$ and recolor the vertex with the first color that is not used by neighbors of S and used by at most one denser neighbor. As we shall see, there are at least $\gamma \cdot \zeta$ such colors with high probability², so $\tilde{O}(\Delta/\zeta)$ color tries suffice with high probability. To verify if a color is available, the algorithm loops through its color class (line 4). Finally, if the color was used by a *unique* denser neighbor w (line 6), we steal the color from w and recolor the denser vertex using **RecolorDense**.

Algorithm 2: For a sparse vertex v

```

1 function RecolorSparse( $v$ ):                                //  $\varphi[v] = \perp$  and  $v \in S$ 
2   Sample  $\chi \in [\Delta + 1]$  and  $w \leftarrow \perp$ 
3   for  $u \in \Phi[\chi]$  do
4     if  $u \in N_S(v)$  or  $(u \in N_{V \setminus S}(v)$  and  $w \neq \perp)$  then go back to line 2
5     else if  $u \in N_{V \setminus S}(v)$  and  $w = \perp$  then  $w \leftarrow u$ 
6   if  $w \neq \perp$  then
7     Color( $w, \perp$ ), Color( $v, \chi$ )                        // steal from  $w$ 
8     if  $matched[w] = \perp$  then RecolorDense( $w$ )
9     else RecolorMatching( $w, matched[w]$ )
10  else Color( $v, \chi$ )

```

²For convenience, if no available color exist, the algorithm loops forever. As we explain later, this occurs with probability $1/\text{poly}(n)$. If one wishes the algorithm to have finite expected update time, the algorithm can be modified to restart a phase if no available color was found after $\tilde{O}(n^{1/3})$ samples.

Denser Vertices. We now explain how we recolor denser *unmatched* vertices. Recall that inliers are such that $a_v \leq 8a_C$ and $e_v \leq 8e_C$ (Definition 3.4). Note that the algorithm does not maintain those sets explicitly, as it can simply test for those inequalities. Importantly, inliers always have colors available in the clique palette by the Accounting Lemma (Lemma 3.9). We therefore color them by computing the set of colors used by external neighbors and searching for an available color in $L(D)$ (line 8).

We color outliers like the sparser vertices (lines 2 to 7): by sampling colors in $[\Delta + 1]$. We allow outliers to steal the color of an *unmatched inlier* and then recolor the inlier instead (line 6).

Algorithm 3: For a denser $v \in D$ (which needs recoloring) and is unmatched

```

1 function RecolorDense( $v$ ):                                //  $\varphi[v] = \perp$  and  $\text{matched}[v] = \perp$ 
2   if  $v \notin I_D$  then                                       // color an outlier
3     Sample  $\chi \in [\Delta + 1]$ 
4     if  $\chi \in M_D$  then go to line 3
5     for  $u \in \Phi[\chi]$  if  $u \in N(v)$  but  $u \notin I_D$  then go to line 3
6     if  $\Phi_D[\chi] \cap I_D = \{u\}$  then Color( $u, \perp$ ), Color( $v, \chi$ ), RecolorDense( $u$ )
7     else Color( $v, \chi$ )
8   else Color( $v, \chi$ ) with any  $\chi \in L(D) \setminus \varphi[E(v)]$            //  $v \in I_D$ 

```

To implement **RecolorDense** efficiently, for each almost-clique D , we maintain

- the set of unused colors $L(D) = [\Delta + 1] \setminus \varphi[D]$, a.k.a. the clique palette,
- the set of vertices $\Phi_D[\chi]$ in D with color χ .

Those data-structures are easy to maintain with $O(\log n)$ worst-case update time because each update affects at most two vertices — thus at most two almost-cliques — and each set changes by at most one element.

Recoloring a Matched Anti-Edge. For a pair $u, v \in D$, **RecolorMatching**(u, v) colors — or recolors — u and v the same. As we cannot ensure they share many available colors in the clique palette, we allow u and v to pick any color that is used by neither a matched vertex (line 4) nor external neighbors (line 5). Since we only need few repeated colors, they always have $\Omega(\Delta)$ colors to choose from and we can find one using $O(\log n)$ random samples. However, when doing so, we might have to — in fact, most likely will — recolor an unmatched vertex (line 8).

RecolorMatching is used whenever matched vertices need to get recolored (line 12 in Algorithm 1) or when the matching size needs to be increased (line 12 in Algorithm 4). To increase the size of the matching, **IncrMatching** needs to find a pairs $\{u, v\}$ of unmatched anti-neighbors to same-color. Since we maintain a matching of size $\Theta(a_D)$ while D contains $\Theta(a_D \Delta)$ and anti-degrees are at most $O(\varepsilon \Delta)$, a random anti-edge in F_D has both endpoints unmatched with constant probability.

4.1. Dense Invariants. We begin by showing that our recoloring algorithm maintains the dense invariants (Lemma 4.3). More interestingly, we show in Lemma 4.4 that when the Matching Invariant is broken by an update, one call to **IncrMatching** suffices to fix it.

Lemma 4.3. *Suppose the invariants hold before a call to **RecolorDense**(v) with $\text{matched}[v] = \perp$ or **RecolorMatching**(u, v) or **RecolorSparse**(v). If the call ends, it maintains the invariants.*

Proof. **TOPROVE 4** ■

Algorithm 4: Same-colors a pair $u, v \in D$

```

1 function RecolorMatching( $u, v$ ):
2   if  $\text{matched}[u] = \text{matched}[v] = \perp$  then  $\text{matched}[u] \leftarrow v, \text{matched}[v] \leftarrow u$ 
3   Sample  $\chi \in [\Delta + 1]$ 
4   if  $\chi \in M_D$  then go to line 3
5   for  $w \in \Phi[\chi]$  if  $w \in E(v) \cup E(u)$  then go to line 3
6    $M_D \leftarrow M_D + \chi$ 
7   if  $\Phi_D[\chi] = \{w\}$  then                                // steal the color of an unmatched vertex
8     |  $\text{Color}(w, \perp), \text{Color}(u, \chi), \text{Color}(v, \chi)$  and  $\text{RecolorDense}(w)$ 
9   else  $\text{Color}(u, \chi)$  and  $\text{Color}(v, \chi)$ 
10 function IncrMatching( $D$ ):
11   Sample  $\{u, v\} \in F_D$  uniformly at random
12   if  $\text{matched}[u] = \text{matched}[v] = \perp$  then  $\text{RecolorMatching}(u, v)$ 
13   else go to line 11

```

Lemma 4.3 shows that **RecolorInsert** and **RecolorDelete** maintain the Dense Balance Invariant since it is only affected when vertices are recolored. On the other hand, the Matching Invariant depends on the values of $|M_D|$ and a_D , which are affected by insertions or deletions of edges.

Lemma 4.4. *Consider a fixed update $\text{RecolorInsert}(u, v)$ or $\text{RecolorDelete}(u, v)$ before which the invariant holds. When $\text{RecolorDelete}(u, v)$ ends or when $\text{RecolorInsert}(u, v)$ reaches line 10, the invariants holds.*

Proof. **TOPROVE 5** ■

4.2. Complexity of Recoloring the Denser Vertices. We investigate the time complexity of our recoloring procedures for dense vertices and express it in terms of the maximal size of a color class at the time of the update and the sparsity parameter ζ . Observe that the probabilistic statements of **Lemmas 4.5 to 4.7** are only over the randomness sampled by the algorithm during the corresponding calls to **RecolorDense**, **RecolorMatching** and **IncrMatching**.

Lemma 4.5. *Suppose that the invariants hold before a call to $\text{RecolorDense}(v)$ where $v \in D$, $\varphi[v] = \perp$ and $\text{matched}[v] = \perp$. W.h.p. over the random colors it samples, the call ends in*

$$O\left(\max_{\chi} |\Phi[\chi]| \log^2 n + \zeta \log n\right)$$

time.

Proof. **TOPROVE 6** ■

RecolorMatching colors u and v the same way **RecolorDense** colors an outlier. The complexity of recoloring an unmatched denser vertex (line 8) is upper bounded by **Lemma 4.5**. So we obtain the following lemma:

Lemma 4.6. *Suppose all invariants hold before a call to $\text{RecolorMatching}(u, v)$. W.h.p., the call ends after $O(\max_{\chi} |\Phi[\chi]| \log^2 n + \zeta \log n)$ time.*

It remains to bound the time spent on fixing the matching size with **IncrMatching**.

Lemma 4.7. *Consider a call to $\text{IncrMatching}(D)$ before which the Dense Balance Invariants holds but not the Matching Invariant. W.h.p., it ends in $O(\max_{\chi} |\Phi[\chi]| \log^2 n + \zeta \log n)$ time and $|M_D|$ has increased by one.*

Proof. **TOPROVE 7** ■

4.3. Recoloring the Sparser Vertices. RefreshColoring ensures vertices of S have sufficiently many colors in their palette at the beginning of the phase to maintain large palettes over entire phases. Importantly, vertices in S conflict only with neighbors in S . Recall that $L_S(v) = [\Delta + 1] \setminus \varphi[N_S(v)]$ is the palette of colors $v \in S$ can adopt with respect to φ .

Lemma 4.8. *Suppose RefreshColoring succeeded. A call to RecolorSparse(v) where $\varphi[v] = \perp$ and $v \in S$ ends in*

$$O\left(\Delta/\zeta \cdot \max_{\chi} |\Phi[\chi]| \log^2 n + \zeta \log n\right)$$

time with high probability over the random colors it samples.

Proof. **TOPROVE 8** ■

Overall, our algorithm is efficient only if we can maintain small enough color classes. The Dense Balance property ensures that denser vertices contribute $O(n/\Delta)$ to each color class. The contribution of vertices in S is accounted for separately, using an argument over the randomness of the entire phase:

Lemma 4.9. *Suppose that RefreshColoring succeeded. With high probability over the randomness of the whole phase, for every $\chi \in [\Delta + 1]$, its color class contains*

$$(1) \quad |\Phi[\chi] \cap S| \leq O\left(\frac{n}{\zeta} + \frac{t}{\zeta} + \log n\right)$$

sparser vertices before every update of the phase.

Proof. **TOPROVE 9** ■

Remark 4.10. *Eq (1) holds with high probability over all the randomness used since the phase began, unlike Lemmas 4.5, 4.6 and 4.8, which only consider the current update. Conditioning on Eq (1) would introduce a bias in Lemmas 4.5, 4.6 and 4.8, which explains why we instead express runtimes in Lemmas 4.5, 4.6 and 4.8 as functions of the color class sizes. Since all events occur with high probability, they hold simultaneously by the union bound.*

4.4. Proof of Theorem 1. Let δ be the universal constant from Lemma 3.5 and define $\varepsilon = 1/110$ and $\zeta = n^{2/3}$. When $\Delta < n^{2/3}/(\varepsilon^2\delta)$, we use the naive algorithm that scans the neighborhood of the vertices after each update. When $\Delta \geq n^{2/3}/(\varepsilon^2\delta)$, we use Algorithm 1. We consider a fixed phase and show that the amortized update time is $\tilde{O}(n^{2/3})$ with high probability. Recall that a phase is a sequence of $t = \gamma \cdot \zeta$ updates, where γ is the universal constant described in Proposition 4.1. Moreover, one should bear in mind that the algorithm of Lemma 3.5 maintains an (ε, ζ) -sparser-denser decomposition with $O(\log n)$ update time such that the vertex partition is fixed throughout the phase.

Correctness & Invariants. RefreshColoring gives a total and proper $(\Delta+1)$ -coloring and neither RecolorInsert or RecolorDelete introduce color conflicts. By induction, the invariants hold w.h.p. before every update of the phase. Indeed, by Proposition 4.1, if RefreshColoring ends, w.h.p., the invariants hold before the first update of the phase. By Lemmas 4.3 and 4.4, invariants are maintained by RecolorDense, RecolorMatching and RecolorSparse, and thus are preserved after each update.

Efficiency. By Proposition 4.1, computing the fresh coloring takes $\tilde{O}(n + n^2/\zeta)$ time. With high probability, RefreshColoring succeeds and henceforth we condition on its success. By

Lemmas 4.5, 4.6 and 4.8, w.h.p., each update ends in $\tilde{O}\left(\frac{\Delta}{\zeta} \max_{\chi} |\Phi[\chi]| + \zeta\right)$ time. On the other hand, w.h.p., before every update and for every $\chi \in [\Delta + 1]$, we have that,

$$|\Phi[\chi]| = |\Phi[\chi] \cap S| + \sum_{i=1}^r |D_i \cap \Phi[\chi]| \leq |\Phi[\chi] \cap S| + \frac{4n}{\Delta} \leq O\left(\frac{n}{\zeta} + \log n\right)$$

because, by the Dense Balance Invariant, each of the at most $2n/\Delta$ almost-cliques contributes at most two to $\Phi[\chi]$, and by **Lemma 4.9** for the sparse vertices. With the union bound, the bounds on the update time and the color class sizes hold with high probability. Overall, w.h.p., the amortized update time during this phase is

$$\tilde{O}\left(\frac{n + n^2/\zeta}{t} + \frac{n\Delta}{\zeta^2} + \zeta\right) = \tilde{O}\left(\frac{n^2}{\zeta^2} + \zeta\right) = \tilde{O}(n^{2/3})$$

for $\zeta = n^{2/3} \leq O(\Delta) \leq O(n)$ and $t = \gamma \cdot \zeta$. \blacksquare

Remark 4.11. *We hide three log-factors in our soft-Oh notation. We made no attempt at optimizing it but believe that using more sophisticated data-structures for sets and more careful concentration arguments, at least two of those could be avoided.*

5. COMPUTING A FRESH COLORING

Recall that every t updates, the algorithm recomputes a fresh coloring with nice properties. In this section, we provide and analyse this algorithm, thus prove the following proposition.

Proposition 4.1. *There exists a universal constant $\gamma \in (0, 1)$ such that the following holds. For $\zeta \geq \Omega(\log n)$ for some large hidden constant, let S, D_1, \dots, D_r be a (ε, ζ) -decomposition of an n -vertex graph with maximum degree Δ . W.h.p., **RefreshColoring** runs in $\tilde{O}(n + n^2/\zeta)$ time, producing a total coloring φ such that*

- (Sparse Slack) every sparse vertex v has $|L_S(v)| \geq 3\gamma \cdot \zeta$,
- (Sparse Balance) $|\Phi[\chi] \cap S| = O(n/\zeta + \log n)$ for every $\chi \in [\Delta + 1]$,
- (Dense Balance) in each D_i , every color is used at most twice, and
- (Matching) each D_i has at least $\lfloor 8a_{D_i} \rfloor$ redundant colors.

RefreshColoring begins by uncoloring every vertex to start from a fresh state. It then uses a procedure called **OneShotColoring** in which every vertex of S tries a random color. It is a well-known fact that the remaining uncolored vertices have “slack” (**Proposition 5.1**). The algorithm colors the remaining vertices in S in a random order, which ensures that the total number of color trials is $\tilde{O}(n)$ [**BRW25**]. Denser vertices do not participate in **OneShotColoring** to ensure that the partial coloring it produced does not hinder the construction of a colorful matching. This is why the Sparse Slack guarantee is only in terms of $L_S(v)$ rather than $L(v) = [\Delta + 1] \setminus \varphi[N(v)]$. **RefreshColoring** colors almost-cliques sequentially, using the coloring primitives devised in **Section 4**.

Generating Slack. To analyse the slack produced by **OneShotColoring**, we use the following well-known results in the distributed graph literature (e.g., [**CLP20**, Lemma 3.3] or [**HKMT21**, Lemma 6.1] or even [**MR02**]). Authors of [**ACK19**, Lemma A.1] and [**BRW25**] use a version of this result for $\zeta \geq \Omega(\varepsilon^2 \Delta)$.

Proposition 5.1. *Suppose G is a graph of maximal degree $\Delta \gg 1$. If φ is the coloring at the end of **OneShotColoring**, then a ζ -sparse vertex v with $\zeta \gg 1$ has that³*

$$|L(v)| \geq [\# \text{uncolored vertices in } N(v)] + 2^{-22} \cdot \zeta$$

³we did not attempt to optimize this constant

Algorithm 5: Computing a Fresh Coloring

```

1 function RefreshColoring:
2   for each  $v$  do Color( $v, \perp$ )                                // uncolor all vertices first
3   OneShotColoring                                           // generating slack
4   Sample a uniform permutation  $\pi$  of  $S$ 
5   for every uncolored  $v \in S$  in the order of  $\pi$  do RecolorSparse( $v$ )
6   for every  $i = 1, 2, \dots, r$  do
7     for  $\lfloor 8a_{D_i} \rfloor$  times do IncrMatching( $D_i$ )
8     for every uncolored  $v \in O_{D_i}$  do ColorDense( $v$ )
9     for every uncolored  $v \in I_{D_i}$  do ColorDense( $v$ )

10 function OneShotColoring:
11   for every  $v \in S$ , with probability  $1/8$  do Color( $v, \chi$ ) with a random  $\chi \in [\Delta + 1]$ 
12   for every  $\chi \in [\Delta + 1]$  and every pair  $u, v \in \Phi[\chi] \cap S$  do
13     if  $\{u, v\} \in E$  then Color( $v, \perp$ ), Color( $u, \perp$ )

14 function ColorDense( $v$ ):
15   Sample  $\chi \in L(C)$ 
16   for  $u \in \Phi[\chi]$  if  $u \in N(v)$  then go to line 15
17   Color( $v, \chi$ )

```

with probability at least $1 - \exp(-\Omega(\zeta))$.

Through [Proposition 5.1](#), OneShotColoring ensures the Sparse Slack property.

Lemma 5.2. *Let ζ and S, D_1, \dots, D_r be as described by [Proposition 4.1](#). With high probability, we have that*

- if RefreshColoring reaches line 6, every vertex in S is colored, the Sparse Slack and Sparse Balance properties hold; and
- before every call to RecolorSparse (line 5), the Sparse Balance property holds.

Proof. [TOPROVE 10](#) ■

As [\[BRW25, Appendix A.1\]](#) showed, this algorithm makes $\tilde{O}(n)$ color trials (executions of line 2 in RecolorSparse) with high probability. We provide a proof of that fact in [Appendix B.1](#) for completeness. We obtain the following corollary by union bound: OneShotColoring runs in $O(n + \Delta \max_{\chi} |\Phi[\chi]|^2) = \tilde{O}(n + n^2/\Delta)$ and each of the $\tilde{O}(n)$ color trials takes $O(n/\zeta + \log n)$ time by [Lemma 5.2](#).

Corollary 5.3. *W.h.p., RefreshColoring reaches line 6 in $\tilde{O}(n + n^2/\zeta)$ time, every vertex in S is colored, the Sparse Slack and Sparse Balance properties hold.*

Coloring Denser Vertices. Random color trials in $[\Delta + 1]$ according to a random ordering could also be used for dense vertices, but as we need to ensure the Matching and Dense Balance invariants, we use a more tailored approach albeit similar.

Lemma 5.4. *Let $\zeta, S, D_1, \dots, D_r$ be as described by [Proposition 4.1](#). For any partial coloring outside of $D = D_i$ such that the Sparse Slack, Sparse Balance and Dense Balance hold, w.h.p., RefreshColoring extends the coloring to D (lines 7 to 9) in $\tilde{O}(n + n\Delta/\zeta)$ time and such that D contains a colorful matching of size $\lfloor 8a_D \rfloor$ and the Dense Balance holds.*

Proof. **TOPROVE 11** ■

Proposition 4.1 follows trivially from combining **Corollary 5.3** and **Lemma 5.4** for all $O(n/\Delta)$ almost-cliques.

Acknowledgment. Discussions with participants of Dagstuhl Seminar 24471, Graph Algorithms: Distributed Meets Dynamic, served as a key motivation for this work.

REFERENCES

- [ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786, 2019.
- [AY25] Sepehr Assadi and Helia Yazdanyar. Simple sublinear algorithms for $(\Delta + 1)$ vertex coloring via asymmetric palette sparsification. In Ioana Oriana Bercea and Rasmus Pagh, editors, *2025 Symposium on Simplicity in Algorithms, SOSA 2025, New Orleans, LA, USA, January 13-15, 2025*, pages 1–8. SIAM, 2025.
- [BCHN18] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In Artur Czumaj, editor, *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.
- [BEPS16] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3):20:1–20:45, 2016.
- [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.
- [BRW25] Soheil Behnezhad, Rajmohan Rajaraman, and Omer Wasim. Fully dynamic $(\Delta + 1)$ -coloring against adaptive adversaries. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4983–5026. SIAM, 2025.
- [CDP21] Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in congested clique and MPC. *SIAM J. Comput.*, 50(5):1603–1626, 2021.
- [CFG⁺19] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\Delta + 1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 471–480. ACM, 2019.
- [CLP20] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. Distributed $(\Delta + 1)$ -coloring via ultrafast graph shattering. *SIAM J. Comput.*, 49(3):497–539, 2020.
- [CMNS24] Yi-Jun Chang, Gopinath Mishra, Hung Thuan Nguyen, and Farrel D. Salim. Round and communication efficient graph coloring. *CoRR*, abs/2412.12589, 2024.
- [Doe20] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*, Natural Computing Series, pages 1–87. Springer, 2020.
- [DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, Cambridge, 2009.
- [EPS15] Michael Elkin, Seth Pettie, and Hsin-Hao Su. $(2\Delta - 1)$ -edge-coloring is much easier than maximal matching in the distributed setting. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 355–370. SIAM, 2015.
- [FGH⁺23] Maxime Flin, Mohsen Ghaffari, Magnús M. Halldórsson, Fabian Kuhn, and Alexandre Nolin. Coloring fast with broadcasts. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023, Orlando, FL, USA, June 17-19, 2023*, pages 455–465. ACM, 2023.
- [FGH⁺24] Maxime Flin, Mohsen Ghaffari, Magnús M. Halldórsson, Fabian Kuhn, and Alexandre Nolin. A distributed palette sparsification theorem. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*. SIAM, 2024.
- [FM24] Maxime Flin and Parth Mittal. $(\Delta + 1)$ vertex coloring in $O(n)$ communication. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 416–424. ACM, 2024.

- [HKMT21] Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. Efficient randomized distributed coloring in CONGEST. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1180–1193. ACM, 2021.
- [HNT21] Magnús M. Halldórsson, Alexandre Nolin, and Tigran Tonoyan. Ultrafast distributed coloring of high degree graphs. *CoRR*, abs/2105.04700, 2021.
- [HP22] Monika Henzinger and Pan Peng. Constant-time dynamic $(\Delta + 1)$ -coloring. *ACM Trans. Algorithms*, 18(2):16:1–16:21, 2022.
- [HSS18] David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\Delta + 1)$ -coloring in sublogarithmic rounds. *J. ACM*, 65(4):19:1–19:21, 2018.
- [KQ21] William Kuszmaul and Qi Qi. The multiplicative version of azuma’s inequality, with an application to contention analysis. *CoRR*, abs/2102.05077, 2021.
- [Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.
- [MR02] Michael Molloy and Bruce Reed. *Graph colouring and the probabilistic method*, volume 23. Springer Science & Business Media, 2002.
- [Ree98] Bruce A. Reed. ω , Δ , and χ . *J. Graph Theory*, 27(4):177–212, 1998.

APPENDIX A. CONCENTRATION INEQUALITIES

We use the Chernoff Bound to prove concentration of random variables around their expected values. To comply with dependencies between our random variables, we use a stronger form than the classic variant of the bound. We refer readers to [DP09] or [Doe20, Section 1.10] or [KQ21, Corollaries 6 and 14].

Lemma A.1 (Chernoff bounds with stochastic domination). *Let X_1, \dots, X_n be random variables and Y_i be any function of the first i variables with value in $[0, 1]$. Consider $Y = \sum_{i=1}^n Y_i$. If there exists $q_1, \dots, q_n \in [0, 1]$ such that $\mathbb{E}[Y_i | X_1, \dots, X_{i-1}] \leq q_i$ for every $i \in [n]$, then for any $\delta > 0$,*

$$(2) \quad \mathbb{P}[Y \geq (1 + \delta)\mu] \leq \exp\left(-\frac{\delta^2}{2 + \delta}\mu\right) \quad \text{where} \quad \mu \geq \sum_{i=1}^n q_i.$$

Conversely, if there exists $q_1, \dots, q_n \in [0, 1]$ such that $\mathbb{E}[Y_i | X_1, \dots, X_{i-1}] \geq q_i$ for every $i \in [n]$, then for any $\delta \in (0, 1)$,

$$(3) \quad \mathbb{P}[Y \leq (1 - \delta)\mu] \leq \exp\left(-\frac{\delta^2}{2}\mu\right) \quad \text{where} \quad \mu \leq \sum_{i=1}^n q_i.$$

APPENDIX B. MISSING PROOFS

B.1. Nearly Linearly Many Color Trials for Fresh Colorings. This is a streamlined adaptation of [BRW25, Appendix A.1]. It was already shown in [FM24] that the *expected* number of trials is $O(n \log n)$ — even when vertices are not necessarily sparse. In fact, our proof does not rely on the sparsity or the slack provided by `OneShotColoring` at all.

Lemma B.1. *The number of color trials (line 2 in `RecolorSparse`) made by `RefreshColoring` is $O(n \log^2 n)$ with high probability.*

Proof. TOPROVE 12 ■

REYKJAVIK UNIVERSITY, ICELAND.

Email address: maximef@ru.is

REYKJAVIK UNIVERSITY, ICELAND.

Email address: mmh@ru.is