Dynamic Edge Coloring with Improved Approximation

Ran Duan * Haoqing He † Tianyi Zhang ‡

Abstract

Given an undirected simple graph $G = (V, E)$ that undergoes edge insertions and deletions, we wish to efficiently maintain an edge coloring with only a few colors. The previous best dynamic algorithm by [3] could deterministically maintain a valid edge coloring using $2\Delta-1$ colors with $O(\log \Delta)$ update time, where Δ stands for the current maximum vertex degree of graph G. In this paper, we first propose a new static $(1+\epsilon)\Delta$ edge coloring algorithm that runs in near-linear time. Based on this static algorithm, we show that there is a randomized dynamic algorithm for this problem that only uses $(1+\epsilon)\Delta$ colors with $O(\log^8 n/\epsilon^4)$ amortized update time when $\Delta \geq \Omega(\log^2 n/\epsilon^2)$, where $\epsilon > 0$ is an arbitrarily small constant.

1 Introduction

Graph edge-coloring is a basic problem in computer science. A valid edge coloring of a simple undirected graph is an assignment of all edges to colors such that no two different edges with the same color share a common endpoint. Clearly, any valid edge coloring must use at least Δ different colors, where Δ denotes the maximum vertex degree in the graph. On the one hand, a cubic time algorithm from [16] could compute a $\Delta+1$ coloring for any graphs; on the other hand, it was proved in [11] that deciding whether the minimum number of colors used by any edge coloring is Δ or $\Delta+1$ is NP-complete.

In this paper we study the edge coloring problem in a dynamic setting; namely the underlying graph undergoes a sequence of edge insertions and deletions, and we wish to efficiently maintain a valid edge coloring using a small number of different colors. This problem was first studied in [2] which gave a dynamic algorithm with $\tilde{O}(\sqrt{\Delta})$ worst-case update time using $O(\Delta)$ colors. This result was significantly improved by [3] which proposed a simple deterministic dynamic algorithm using $2\Delta - 1$ colors with $O(\log \Delta)$ worst-case update time. As we can see from the literature, so far there is no dynamic algorithm that could efficiently maintain a valid edge coloring using $2\Delta - 2$ colors. The essential difference between $2\Delta - 1$ coloring and $2\Delta - 2$ coloring is that the former problem is locally-fixable [3] while the latter is not. Local fixability usually makes a dynamic problem easier because one can always recover the output without changing much of the current structure.

So what do we know about edge coloring with less than $2\Delta - 1$ colors? A negative result from [4] shows that, for any $c \leq \Delta/3$, there exists a $(\Delta + c)$ partially colored graph with a unique uncolored edge, such that in order to obtain a full $(\Delta + c)$ edge coloring of the entire graph, one always needs to change the colors of $\Omega(\frac{\Delta}{c} \log n)$ many different edges. Therefore, it seems rather hard to look for dynamic algorithms with $\text{poly}(\log n, 1/\epsilon)$ update time using, say, $\Delta + \sqrt{\Delta}$ colors, because of a lower bound of $\Omega(\sqrt{\Delta} \log n)$ which is large when Δ is polynomially large. Given this, it would be natural to consider edge coloring using at most $(1+\epsilon)\Delta$ colors, where $\epsilon \in (0, 1)$ is a constant value.

Problem 1. *Is there a dynamic algorithm that efficiently maintains a $(1 + \epsilon)\Delta$ edge coloring?*

Our results We have made a partial progress to answer Problem 1 by proposing a dynamic algorithm that maintains a $(1 + \epsilon)\Delta$ edge coloring with poly-logarithmic amortized update time but only for $\Delta \geq \Omega(\log^2 n/\epsilon^2)$. The main idea of our dynamic algorithm originates from a new near-linear time algorithm for the static version of $(1 + \epsilon)\Delta$ edge coloring, which is a byproduct of this paper.

Theorem 1.1. *For any constant $\epsilon > 0$, there exists a randomized algorithm that finds a $(1+\epsilon)\Delta$ edge coloring for a simple undirected graph G (in the static setting) in $O(m \log^6 n/\epsilon^2)$ time, where $\Delta = \Omega(\log n/\epsilon)$ is the maximum vertex degree.*

This is the first near-linear time algorithm for edge coloring in general graphs using $(1 + \epsilon)\Delta$ colors with $\epsilon < 0.5$. Previously we only know about two near-linear time algorithms for bipartite graphs [1, 5] using exactly Δ colors, and by a reduction from edge coloring in general graphs to bipartite graphs, these algorithms yield a $3\lceil\Delta/2\rceil$[1] edge coloring algorithm for general

*Institute for Interdisciplinary Information Sciences, Tsinghua University, duanran@mail.tsinghua.edu.cn

†Institute for Interdisciplinary Information Sciences, Tsinghua University, hehq13@mails.tsinghua.edu.cn

‡Institute for Interdisciplinary Information Sciences, Tsinghua University, tianyi-z16@mails.tsinghua.edu.cn

[1] We thank an anonymous reviewer for pointing out this result.

graphs in near-linear time [12]. For general graphs with $\Delta + 1$ colors, the well-known Vizing's algorithm [16] gives an $O(mn)$ time upper bound, which was improved to $\tilde{O}(\Delta m)$ and $\tilde{O}(mn^{1/2})$ in [9], still far from being near-linear when Δ is polynomially large.

Next we state our main results for dynamic edge coloring with an improved approximation for large values of Δ with poly-logarithmic update time.

THEOREM 1.2. *For any constant $\epsilon > 0$, there is a randomized fully dynamic algorithm that maintains a $(1+\epsilon)\Delta$ edge coloring of an undirected simple graph in **worst-case** $O(\log^7 n/\epsilon^2)$ update time, where $\Delta = \Omega(\log^2 n/\epsilon^2)$ denotes an upper bound on the maximum vertex degree and is **a fixed value** throughout edge insertions and deletions.*

THEOREM 1.3. *For any constant $\epsilon > 0$, there is a randomized fully dynamic algorithm that maintains a $(1+\epsilon)\Delta$ edge coloring of an undirected simple graph in **amortized** $O(\log^8 n/\epsilon^4)$ update time, where $\Delta = \Omega(\log^2 n/\epsilon^2)$ denotes the maximum vertex degree and could possibly **change over time**.*

As we will see, Theorem 1.3 is achieved by applying Theorem 1.2 as a black box to deal with a changing parameter Δ, but with a cost of turning a worst-case update time into an amortized update time.

Related work Dynamic edge coloring is a relatively new problem in the field of dynamic graph algorithms. Other than the two papers [3, 2] we have mentioned, there are only a few experimental results [7, 13, 15, 10].

Another line of work focuses on edge coloring in the LOCAL model. Here is a brief overview of distributed algorithms on edge coloring using less than $2\Delta - 1$ colors. All known LOCAL algorithms are randomized, expect for the $(\Delta + 1)$ coloring from [16] which takes time proportional to the graph diameter. When one is allowed to use 1.6Δ colors, there is a randomized algorithm by [14] with $O(\log n)$ running time that works for all $\Delta > \log^{1+o(1)} n$. As for $(1+\epsilon)\Delta$ edge coloring, the earliest result from [6] uses $O(\epsilon^{-1}\log\epsilon^{-1} + \log n)$ time under the condition that $\Delta > \log^{1+o(1)} n$. The range of Δ was extended to $\Delta > \Delta_\epsilon$ in [8] along with an algorithm with running time $O((\epsilon^{-2}\log\epsilon^{-1} + \log^* n)\lceil\frac{\log n}{\epsilon^2\Delta^{1-o(1)}}\rceil)$, where Δ_ϵ is a constant that only depends on ϵ. The running time was significantly improved by [4] to $O(\log\epsilon^{-1}\lceil\frac{\log n}{\epsilon^2\Delta^{1-o(1)}}\rceil + \log^* n)$ and $O(\log\epsilon^{-1}\lceil\frac{\log n}{\epsilon^2\Delta^{1-o(1)}}\rceil + (\log\log n)^{3+o(1)})$ that works for all $\Delta > (\log n)^{1+o(1)}/\epsilon$ and $\Delta > \Delta_\epsilon$ respectively. To go beyond $(1+\epsilon)\Delta$ colors, [4] also proposed an algorithm with $O(\log\Delta\lceil\frac{\log n}{\epsilon^2\Delta^{1-o(1)}}\rceil + (\log\log n)^{3+o(1)})$ time using $\Delta + \tilde{O}(\sqrt{\Delta})$ colors, which works for any value of Δ.

2 Preliminaries

Let $G = (V, E)$ be the undirected simple graph of interest, with $n = |V|$ and $m = |E|$. Denote by Δ an upper bound on the maximum vertex degree of G. Let $\mathcal{S} = [(1+\epsilon)\Delta]$ be the set of all available colors, where $0 < \epsilon < 1$ is a constant. Throughout the execution of our algorithms (static and dynamic), let $\mathsf{EdgeColor} : E \to \mathcal{S} \cup \{\perp\}$ be the current coloring of edges by our algorithms. For any edge $e \in E$, when $\mathsf{EdgeColor}[e] = i \in [(1+\epsilon)\Delta]$, it means e is colored with the i-th color; when $\mathsf{EdgeColor}[e] = \perp$, it means e is not colored yet, and thus $\mathsf{EdgeColor}$ would be a partial edge coloring of graph G. For any vertex $v \in V$, define set $\mathsf{UsedColor}[v] = \{\mathsf{EdgeColor}[e] \mid e = (v, u), u \in V\}$ be the set of colors (including $\perp$) around v. Finally, for any partially colored graph G and any subset $U \subseteq V$, let $G[U]$ be the induced subgraph of G on vertex subset U with the same partial coloring.

Next we discuss some basic tools which will be utilized in our static and dynamic algorithms. Consider any partially colored graph G. First we introduce a new notion of "palette" which refers to a proper subset of colors.

DEFINITION 2.1. (PALETTE) *For a partially colored G, a color subset $C \subseteq \mathcal{S}$ is a palette, if for every vertex $v \in V$, $C \cap (\mathcal{S} \setminus \mathsf{UsedColor}[v]) \neq \emptyset$; in words, there is a color from C which is not present around v.*

The following lemma shows an easy construction of palettes using randomization.

LEMMA 2.2. *Let G be a partially colored graph, and let B be a positive constant integer. Then, as long as $\Delta \geq \Omega(B\log n/\epsilon)$ a random subset of colors $C \subset \mathcal{S}$ of size $\lceil B\log n/\epsilon\rceil$ makes a valid palette with respect to the partial coloring, with probability $1 - n^{-B/2+1}$.*

Proof. Because of a uniform upper bound of Δ on vertex degrees, for each vertex v, at least a proportion of $\epsilon/(1+\epsilon)$ colors are missing from its incident edges. Hence a random color $c \in \mathcal{S}$ comes from $\mathcal{S} \setminus \mathsf{UsedColor}[v]$ with probability at least $\epsilon/(1+\epsilon)$. Therefore, the probability that $C \cap (\mathcal{S} \setminus \mathsf{UsedColor}[v]) = \emptyset$ is smaller than $(1-\epsilon/(1+\epsilon))^{B\log n/\epsilon} < (1-\epsilon/2)^{B\log n/\epsilon} \leq n^{-B/2}$. By the union bound, with probability $\geq 1 - n^{-B/2+1}$, we have $C \cap (\mathcal{S} \setminus \mathsf{UsedColor}[v]) \neq \emptyset$ for all $v \in V$.

Another notion we heavily rely on is called "alternating path" which is defined below.

DEFINITION 2.3. (ALTERNATING PATH) *Let G be a partially colored graph. Let $c_1, c_2 \in \mathcal{S}$ be a pair of different colors. An (c_1, c_2)-alternating path in G is a simple path $u_1 \to u_2 \to \cdots \to u_l$, such that: (1)*

$c_2 \notin \mathsf{UsedColor}[u_1], \{c_1, c_2\} \setminus \mathsf{UsedColor}[u_l] \neq \emptyset$; *(2)* $\mathsf{EdgeColor}[(u_{2i-1}, u_{2i})] = c_1$, $\mathsf{EdgeColor}[(u_{2i}, u_{2i+1})] = c_2, \forall i$.

These lemmas below should be folklore regarding alternating paths.

LEMMA 2.4. (VERTEX-DISJOINTNESS) *In any partially colored graph G, for any pair of different colors c_1, c_2, all different (c_1, c_2)-alternating paths are vertex-disjoint or refer to the same simple path but in reverse order.*

LEMMA 2.5. *Consider a **flip** operation: given any (c_1, c_2)-alternating path denoted by $u_1 \to u_2 \to \cdots \to u_l$, we change the coloring $\mathsf{EdgeColor}[(u_{2i-1}, u_{2i})] \leftarrow c_2, \mathsf{EdgeColor}[(u_{2i}, u_{2i+1})] \leftarrow c_1, \forall i$. Then after this operation, this path becomes a (c_2, c_1)-alternating path, and the partial coloring stays valid; plus, $\mathsf{UsedColor}[u_1] \leftarrow \mathsf{UsedColor}[u_1] \oplus \{c_1, c_2\}$[2] and $\mathsf{UsedColor}[u_l] \leftarrow \mathsf{UsedColor}[u_l] \oplus \{c_1, c_2\}$, and all other sets $\mathsf{UsedColor}[v], v \notin \{u_1, u_l\}$ stay unchanged.*

3 Static edge coloring in near-linear time

In this section we introduce a randomized near-linear time algorithm for approximate edge coloring; that is, we prove Theorem 1.1.

In the following lemma, we review an algorithm that, given a palette, one can color any currently uncolored edge by running Vizing's algorithm only using colors from this palette.

LEMMA 3.1. (A SLIGHT REVISION OF [16]) *Let (u, v) be an uncolored edge in G, and let C be a palette of size $O(\log n/\epsilon)$ with respect to the current partial coloring of G. Then we can adjust the part of G with colors only from C so that (u, v) can be colored using a color from C.*

Proof. We compute a chain of vertices around u in the following manner. Start from $v_0 = v$, and $k = 0$. By Definition 2.1, there exists a color $c_0 \in C \cap (\mathcal{S} \setminus \mathsf{UsedColor}[v_0])$. Suppose we already have two sequences of different neighbors of u and colors which are $v_0, v_1, v_2, \cdots, v_k$ and $c_0, c_1, \cdots, c_k$, such that: (1) $\mathsf{EdgeColor}[(u, v_i)] = c_{i-1}, \forall 1 \leq i \leq k$; (2) $c_i \in C \cap (\mathcal{S} \setminus \mathsf{UsedColor}[v_i]), \forall 0 \leq i \leq k$. Then consider two cases below.

- If c_k is also not present around u, namely $c_k \notin \mathsf{UsedColor}[u]$, then we simply rotate the colors: $\mathsf{EdgeColor}[(u, v_0)] \leftarrow c_0, \mathsf{EdgeColor}[(u, v_1)] \leftarrow c_1, \cdots, \mathsf{EdgeColor}[(u, v_k)] \leftarrow c_k$. This operation would obtain a new partially coloring where $\mathsf{EdgeColor}[(u, v)] \neq \perp$.
- Else, there exists a unique neighbor w of u such that (u, w) has color c_k. If w has not appeared before among $v_0, v_1, v_2, \cdots, v_k$, then we set $v_{k+1} = w$ and increment $k \leftarrow k+1$; otherwise we do the following things.

  Assume $w = v_j$ for a $j \in (0, k)$, and then $c_{j-1} = c_k$. Choose an arbitrary $c \in C \cap (\mathcal{S} \setminus \mathsf{UsedColor}[u])$. Find the (c_k, c)-alternating path starting at u. Consider two possible cases of this alternating path.

  (1) The path does not end at v_{j-1}. Then we change the coloring: $\mathsf{EdgeColor}[(u, v_0)] \leftarrow c_0, \mathsf{EdgeColor}[(u, v_1)] \leftarrow c_1, \cdots, \mathsf{EdgeColor}[(u, v_{j-1})] \leftarrow c_{j-1}$, and then flip the coloring of the alternating path.

  By Lemma 2.5, only endpoints of the alternating path change their $\mathsf{UsedColor}$ sets. Since v_{j-1} is not one of the endpoints, assigning $\mathsf{UsedColor}[(u, v_{j-1})] \leftarrow c_{j-1}$ is legal. Also note that this ends up with a valid partial coloring with one more edge being colored, which is (u, v_0).

  (2) The path ends at v_{j-1}. Then we first flip the coloring of the alternating path, and modify the coloring: $\mathsf{EdgeColor}[(u, v_i)] \leftarrow \mathsf{EdgeColor}[(u, v_{i+1})], \forall 0 \leq i < k$, and also $\mathsf{EdgeColor}[(u, v_k)] \leftarrow c_k$.

  Since before the modifications $c_{j-1} \notin \mathsf{UsedColor}[v_{j-1}]$, the alternating path must end at v_{j-1} with a c-color edge. By Lemma 2.5, after such modifications, we know $c \notin \mathsf{UsedColor}[v_{j-1}], c_k \notin \mathsf{UsedColor}[u]$, and hence the final coloring is valid.

Pseudocode 1 below summarizes the algorithm described in Lemma 3.1.

COROLLARY 3.1. *Define L to be the length of the (c_1, c_2)-alternating path described in the proof of Lemma 3.1. Then Algorithm 1 can be implemented in $O(L \log n + \log^3 n/\epsilon^2)$ time.*

Proof. Finding each c_k can be done in time $O(\log^2 n/\epsilon)$ by going over all choices in $c_k \in C$ and checking if $c_k \in \mathcal{S} \setminus \mathsf{UsedColor}[v_k]$ using a binary search tree. Also, using binary search trees to index neighbors of vertices by colors, the algorithm runs in time $O(L \log n + k \log^2 n/\epsilon)$, if the algorithm ends up with a sequence of neighbors $v_0, v_1, \cdots, v_k$. Now by the way the algorithm is defined, all colors $c_0, c_1, \cdots, c_{k-1}$ are different and are from the palette C, and therefore $k \leq |C| = O(\log n/\epsilon)$.

Now we turn to describe our main algorithm for the static edge coloring problem. To efficiently color one

[2] $\oplus$ denotes symmetric difference between two sets.

Algorithm 1: Vizing(u, v, C)

```
v_0 ← v, k = 0;
while true do
    select c_k ∈ C ∩ (S \ UsedColor[v_k]);
    if c_k ∉ UsedColor[u] then
        EdgeColor[(u, v_i)] ← EdgeColor[(u, v_{i+1})], ∀0 ≤ i < k;
        EdgeColor[(u, v_k)] ← c_k;
        return;
    else
        find w such that EdgeColor[(u, w)] = c_k;
        if w ∉ v_1, ··· , v_{k−1} then
            k ← k + 1;
            v_k ← w;
        else
            assume w = v_j, j ∈ (0, k);
            select c ∈ C ∩ (S \ UsedColor[u]);
            find (c_k, c)-alternating path ρ starting at u;
            if ρ does not end at v_{j−1} then
                EdgeColor[(u, v_i)] ← EdgeColor[(u, v_{i+1})], ∀0 ≤ i < j;
                flip the coloring of ρ;
            else
                flip the coloring of ρ;
                EdgeColor[(u, v_i)] ← EdgeColor[(u, v_{i+1})], ∀0 ≤ i < k;
                EdgeColor[(u, v_k)] ← c_k;
                return;
```

more edge in a partially colored graph, we might not want to pick an arbitrary edge and apply Algorithm 1, since the length of the alternating path could be as large as $\Omega(n)$. The following lemma says that one can always select an easy edge (u, v) so that L becomes small.

LEMMA 3.2. *Let G be a partially colored graph, and let U be a vertex cover of all uncolored edge. Suppose we can uniform-randomly take samples from U. Then there is a randomized algorithm with running time $O(\frac{n \log^4 n}{\epsilon^2 |U|})$ that colors **one more** uncolored edge in G with probability $\geq 1 - n^{-9}$.*

Proof. We are only interested in the case where $|U| \geq \Omega(\log^3 n/\epsilon^2)$, since otherwise we directly run Algorithm 1 on an arbitrary uncolored edge. Let $u \in U$ be any vertex. By definition there exists $v \in V$ such that (u, v) currently has no color (break ties in an arbitrary way). Then define $(c_1[u], c_2[u])$ to be the ordered pair of colors from C of the alternating path if we choose to run Algorithm 1 for the uncolored edge (u, v) with palette C, and let $\rho[u]$ be the corresponding alternating path itself; if the algorithm does not take any alternating path during its execution, then we simply define $(c_1[u], c_2[u]) = (\perp, \perp)$, and $\rho[u]$ refers to an empty path.

CLAIM 3.3. *All $\rho[u], \forall u \in U$ exist simultaneously in the current graph G.*

Proof of claim. This is because Algorithm 1 does not change any color before it tries to find the alternating path.

Our algorithm is fairly simple. We first take a random palette of size $20 \log n/\epsilon$, and then uniform-randomly take a vertex $u \in U$, as well as a neighbor v such that EdgeColor$[(u, v)] = \perp$. After that we run Algorithm 1 on the uncolored edge (u, v); if the alternating path $\rho[u]$ has length $\geq \frac{3200 n \log^2 n}{\epsilon^2 |U|}$, then we abort and start over. The following pseudo-code OneMore summarizes this procedure.

By Corollary 3.1, the running time of OneMore is $O(\frac{n \log^4 n}{\epsilon^2 |U|})$. Next we analyze its correctness.

The total number of possible pair $(c_1[u], c_2[u])$ is $|C|(|C| - 1) + 1 < 400 \log^2 n/\epsilon^2$. Define $D \subseteq C^2 \cup \{(\perp, \perp)\}$ such that for any $(c_1, c_2) \in D$, there exists at least $\frac{\epsilon^2 |U|}{800 \log^2 n}$ different $u \in U$ with $(c_1[u], c_2[u]) = (c_1, c_2)$. We call u "good" if $(c_1[u], c_2[u]) \in D$.

CLAIM 3.4. *At least half of vertices from U are good.*

Proof of claim. By definition, for any $(c_1, c_2) \notin D$, there are at most $\frac{\epsilon^2 |U|}{800 \log^2 n}$ many $u \in U$ such that $(c_1[u], c_2[u]) = (c_1, c_2)$. Ranging over all different

Algorithm 2: OneMore(G, U)

1 **for** $t = 1, 2, \cdots, 30 \log n$ **do**
2 sample a palette $C \subseteq \mathcal{S}$ of size $20 \log n/\epsilon$;
3 uniform-randomly sample a vertex $u \in U$, and let v be its neighbor such that EdgeColor$[(u, v)] = \perp$;
4 run Vizing(u, v, C) until it discovers $|\rho[u]| > \frac{3200n \log^2 n}{\epsilon^2 |U|}$;
5 **if** *length of* $\rho[u]$ *exceeds* $\frac{3200n \log^2 n}{\epsilon^2 |U|}$ **then**
6 abort the execution of Vizing(u, v, C), and rollback all changes in the coloring;
7 **else**
8 (u, v) is guaranteed to get a color from C and **break**;

Algorithm 3: Color$(G = (V, E))$

1 $i = \log \Delta$;
2 **while** $i \geq 0$ **do**
3 build a uniform-random sampler on vertex set V_i;
4 **while** $G[V_i \cup V_{i+1} \cdots \cup V_{\log \Delta}]$ *is not fully colored* **do**
5 invoke OneMore$(G[V_i \cup V_{i+1} \cdots \cup V_{\log \Delta}], U)$, where $U \subseteq V_i$ is the set of all vertices in V_i with incident uncolored edges;
6 $i \leftarrow i - 1$;

choices of such $(c_1, c_2) \notin D$, the total number of $u \in U$ with $(c_1[u], c_2[u]) = (c_1, c_2)$ is at most $\frac{\epsilon^2 |U|}{800 \log^2 n} \cdot 400 \log^2 n/\epsilon^2 = |U|/2$.

Claim 3.5. *Let $A \subseteq U$ be a set of all good vertices with an equal value of $(c_1[\cdot], c_2[\cdot])$. Then, for at least half of $u \in A$, $|\rho[u]| \leq \frac{3200n \log^2 n}{\epsilon^2 |U|}$.*

Proof of claim. Suppose otherwise; that is, for more than half of $u \in A$, $|\rho[u]| > \frac{3200n \log^2 n}{\epsilon^2 |U|}$. Since at most two different $u, v \in A$ can share the same (c_1, c_2)-alternating path (that is, $\rho[u]$ and $\rho[v]$ refer to the same simple path in reverse order), by Lemma 2.4 the union of all $\rho[u], u \in A$ would have more than $\frac{1}{2} \cdot \frac{|A|}{2} \cdot \frac{3200n \log^2 n}{\epsilon^2 |U|} \geq n$ vertices, contradiction.

By the above two claims, with probability $\geq 1/2$, line-3 of OneMore takes a good $u \in U$; conditioned on that, with probability $\geq 1/2$, line-4 is running on an instance with $|\rho[u]| \leq \frac{3200n \log^2 n}{\epsilon^2 |U|}$. Therefore, the failure rate of every iteration over t is less than $\frac{3}{4} + n^{-9}$, and thus the overall success rate would be at least $1 - (\frac{3}{4} + n^{-9})^{30 \log n} > 1 - n^{-9}$

Finally we show how to repeatedly apply Lemma 3.2 to prove Theorem 1.1.

Proof of Theorem 1.1. For every $i \leq \log \Delta$, let $V_i = \{u \mid \deg(u) \in [2^i, 2^{i+1})\}$. Consider the following algorithm Color. Correctness of coloring is guaranteed by Lemma 3.2. Next we focus on the running time.

Fix one iteration of the outer loop. Let $n_i = |V_i|$, and then by definition $n_i + n_{i+1} + \cdots + n_{\log \Delta} \leq m/2^{i-1}$. By Lemma 3.2, each invocation of OneMore takes time $O(\frac{\log^4 n}{\epsilon^2}(n_i + n_{i+1} + \cdots + n_{\log \Delta})/h) \leq O(\frac{m \log^4 n}{\epsilon^2 2^i h})$, where we assume h denote the current number of vertices in V_i with uncolored incident edges. To make the summation of such terms $O(\frac{m \log^4 n}{\epsilon^2 2^i h})$ as large as possible, we should consider the worst-case scenario where all uncolored edges incident on the same vertex from V_i are colored by consecutive invocations of OneMore. More specifically, suppose for every $1 \leq j \leq n_i$, there are x_j invocations of OneMore with $|U| = j$. Then the running time is proportional to:

$$\sum_{j=1}^{n_i} \frac{m \log^4 n}{\epsilon^2 2^i j} \cdot x_j = \frac{m \log^4 n}{\epsilon^2 2^i} \sum_{j=1}^{n_i} \frac{x_j}{j}$$

$$\leq \frac{m \log^4 n}{\epsilon^2 2^i} \cdot \Big(\frac{1}{n_i} \sum_{j=1}^{n_i} x_j + \sum_{j=1}^{n_i} \frac{1}{j(j+1)} \sum_{k=1}^{j} x_k\Big)$$

$$\leq \frac{m \log^4 n}{\epsilon^2 2^i} \cdot (2^{i+1} + \sum_{j=1}^{n_i} \frac{1}{j(j+1)} \cdot j \cdot 2^{i+1})$$

$$\leq O(m \log^5 n/\epsilon^2)$$

Here the first inequality is by Abel transformation, and the second is by $\sum_{k=1}^{j} x_k \leq j \cdot 2^{i+1}$. Therefore, the aggregate running time summed over all outer loop iterations becomes $\sum_{i=0}^{\log \Delta} O(m \cdot \log^5 n/\epsilon^2) = O(m \log^6 n/\epsilon^2)$, which concludes the proof.

4 Dynamic edge-coloring with a fixed Δ

In this section we will prove Theorem 1.2. To put an emphasis on the conditions of Theorem 1.2, we assume $\Delta = \Omega(\log^3 n/\epsilon^2)$ and is a fixed value throughout the entire sequence of edge insertions and deletions. As we will see, for now we don't deal with a changing Δ because when Δ decreases, lots of colors might be illegal and a large proportion of the graph coloring might be invalidated.

Translation of the uncolored edge We will only be focusing on edge insertions, since when an edge gets

deleted we do not modify the current edge coloring. The rough idea is that, if the alternating path found by Algorithm 1 is very long, then we can translate the uncolored edge to many possible places. After that we lower bound the probability that this translation could bring this edge to a new position where the alternating path is short. Next we formalize the idea of edge translation.

DEFINITION 4.1. *Let $G = (V, E)$ be a partially colored graph where $e \in E$ is the unique uncolored edge, and let C be a palette with respect to G. For another edge $f \in E \setminus \{e\}$, we say e can be* translated *to f with palette C if we can rewrite part of the current coloring of G only with colors from C such that f becomes the unique uncolored edge in G; here "only with colors from C" means we only modify the current edges with colors from C, and we only use colors from C.*

LEMMA 4.2. *Let (u, v) be the unique uncolored edge in G. Fix a palette C. Suppose* Vizing(u, v, C) *has found an alternating path of length L. Then, for any odd integer $1 \le l \le L$, (u, v) can be translated to the l-th edge on the alternating path in $O(l \log n + \log^3 n/\epsilon^2)$ time. Plus, the translation only uses colors from palette C.*

Proof. We follow the notations defined in the proof of Lemma 3.1. Let $(u =)w_1 \to w_2 \to w_3 \to \cdots \to w_L \to w_{L+1}$ denote the (c_k, c)-alternating path found during the execution of Vizing(u, v, C). For any odd integer $l \in [L]$, we next demonstrate how to translate (u, v) to (w_l, w_{l+1}). Let $(v =)v_0, v_1, v_2, \cdots, v_j, \cdots, v_k$ be the sequence of vertices defined in Algorithm 1. First we do the rotation: EdgeColor$[(u, v_0)] \leftarrow$ EdgeColor$[(u, v_1)]$, EdgeColor$[(u, v_1)] \leftarrow$ EdgeColor$[(u, v_2)]$, $\cdots\cdots$, EdgeColor$[(u, v_{j-1})] \leftarrow$ EdgeColor$[(u, v_j)]$, and EdgeColor$[(u, v_j)] \leftarrow \perp$; this operation preserves the validity of edge coloring so far, plus it translates the unique uncolored edge from e to $(u, v_j) = (w_1, w_2)$. This would prove the statement for $l = 1$.

For a general odd integer $1 < l \le L$, we reassign the colors: EdgeColor$[(w_1, w_2)] \leftarrow c$, EdgeColor$[(w_2, w_3)] \leftarrow c_k$, EdgeColor$[(w_3, w_4)] \leftarrow c$, $\cdots\cdots$, EdgeColor$[(w_{l-1}, w_l)] \leftarrow c_k$, and finally EdgeColor$[(w_l, w_{l+1})] \leftarrow \perp$. Note that this operation also preserves the validity of edge coloring because: (1) as $c \notin$ UsedColor$[u]$ right before, EdgeColor$[(w_1, w_2)] \leftarrow c$ does not make a violation; (2) all UsedColor$[w_j], 2 \le j \le l-1$ do not change.

This algorithm only involves at most $l + \log n/\epsilon$ edges, and thus the running time is bounded by $O(l \log n + \log^3 n/\epsilon^2)$; also one can easily verify that we are only dealing with colors from palette C.

Main algorithm We set out to describe our main algorithm that handles edge insertions. For the rest of this section, let $h = \log^{O(1)} n/\epsilon^{O(1)}$ and $l = O(\log n)$ be two parameters to be specified later. Let (u, v) be a newly inserted edge. Roughly speaking, the algorithm iterates over l rounds. In each round, we tentatively apply Algorithm 1 to the current unique uncolored edge (which is initially (u, v)). If Algorithm 1 finds an alternating path of length $\le 2h$, then it can color this uncolored edge in time $O(h \log n)$ by Corollary 3.1. Otherwise, by Lemma 4.2, the current uncolored edge can be translated to any one of the first h edges on the alternating path; in this case, we randomly pick one of those edges as destination and translate the current uncolored edge. Finally we remove all colors in the palette selected in this round from $\mathcal{S}$. Pseudo-code Insert summarizes this algorithm.

Algorithm 4: Insert(u, v)

```
1  X ← S;
2  (x, y) ← (u, v);
3  for t = 1, 2, ··· , l do
4      sample a palette C ⊂ X of size 20 log n/ε;
5      start to run Vizing(x, y, C);
6      if Vizing(x, y, C) has found an alternating
         path of length ≤ 2h then
7          wait until Vizing(x, y, C) finishes, and
             return;
8      else
9          apply Lemma 4.2 to (x, y) with respect
             to palette C, and translate (x, y) to the
             (2k − 1)-th edge (z, w) on the
             alternating path, where 1 ≤ k ≤ h/2 is
             picked uniformly at random;
10         (x, y) ← (z, w);
11         X ← X \ C;
12 return;
```

Clearly its running time is bounded by $O(hl \log n)$. One can see that by the end of Algorithm 4, G becomes fully colored if and only if it returns on line-7.

Success rate analysis We turn to analyze the success rate of an execution of Insert(u, v). For every $1 \le t \le l$, let C_t be the palette sampled in the t-th iteration. Since the choices of $C_1, C_2, \cdots C_l$ are independent of our coloring scheme, we can assume they are all fixed right from the beginning.

LEMMA 4.3. *With probability $\ge 1 - l \cdot n^{-4} \ge 1 - O(n^{-4} \log n)$, C_t is a palette with respect to the current partially colored graph G, $\forall 1 \le t \le l$.*

Proof. The size of the remaining colors $|\mathcal{X}| = |\mathcal{S} \setminus \bigcup_{i=1}^{t-1} C_i| > (1+\epsilon)\Delta - 10l\log n/\epsilon > (1+\epsilon/2)\Delta$ as $\Delta \geq \Omega(\log^2 n/\epsilon^2)$. Hence, by Lemma 2.2 with $B = 10$, the probability that C_t is a palette is at least $1 - n^{-4}$, and the statement follows from a union bound over all $1 \leq t \leq l$.

We construct a probability tree $\mathcal{T}$ of depth l to analyze the success rate. Roughly speaking, every instance of Algorithm 4 can be viewed as a path that starts at the root of $\mathcal{T}$, and in each iteration of the for-loop, it selects a child node according to the randomness used on line-9 and travels downward. Next we formally define the construction of $\mathcal{T}$. Let $\mathcal{T}$ be a tree where each node $p \in \mathcal{T}$ is associated with three fields:

- a type value $\text{Type}[p] \in \{1, 2, 3\}$,
- a probability mass $\mu[p] \in [0, 1]$,
- a set $E[p]$ of vertex-disjoint edges, which is non-empty only when $\text{Type}[p] = 3$.

Initially, for the root r of $\mathcal{T}$, we simply set $\text{Type}[r] = 3$, $E[r] = \{(u, v)\}$, $\mu[r] = 1$. Now, given any node p, we recursively specify its children nodes if $\text{Type}[p] = 3$. Suppose p has depth $t-1 \in [0, l-1]$ on $\mathcal{T}$. Then, for every edge $e = (x, y) \in E[p]$, imagine we remove its current color from G_0 and run $\mathsf{Vizing}(x, y, C_t)$ on G_0. Let $(c_1[e], c_2[e]) \in C_t^2 \cup \{(\perp, \perp)\}$ be the type of the alternating path found during the execution of $\mathsf{Vizing}(x, y, C_t)$ and define $\rho[e]$ to be the $(c_1[e], c_2[e])$-alternating path itself; note that, as before, if no alternating path is ever found, then $(c_1[e], c_2[e]) = (\perp, \perp)$ and $\rho[e]$ is an empty path. Next we create three types of children.

(i) Create a child node q_1 of p, and define $E_1 = \{e \in E[p] \mid |\rho[e]| \leq 2h\}$, and then assign $\text{Type}[q_1] = 1$ and $\mu[q_1] = \mu[p] \cdot \frac{|E_1|}{|E[p]|}$.

(ii) Collect all pairs of $(c_1, c_2) \in C_t^2$, such that $\{e \in E[p] \setminus E_1 \mid (c_1[e], c_2[e]) = (c_1, c_2)\}$ has less or equal to $\frac{\epsilon^2}{\log^3 n}|E[p] \setminus E_1|$ elements. Let D be the set of all such (c_1, c_2)'s. Create a child node q_2 with $\text{Type}[q_2] = 2$, and define E_2 to be the set of $e \in E[p] \setminus E_1$ such that $(c_1[e], c_2[e]) \in D$, and then assign $\mu[q_2] = \mu[p] \cdot \frac{|E_2|}{|E[p]|}$.

(iii) For every $(c_1, c_2) \in C_t^2 \setminus D$, create a child node q_{c_1,c_2}, and define $E_3[c_1, c_2] = \{e \in E[p] \setminus E_1 \mid (c_1[e], c_2[e]) = (c_1, c_2)\}$, and set $\text{Type}[q_{c_1,c_2}] = 3$, $\mu[q_{c_1,c_2}] = \mu[p] \cdot \frac{|E_3[c_1,c_2]|}{|E[p]|}$. After that, for every $e \in E_3[c_1, c_2]$ and every $1 \leq k \leq h/2$, collect the $(2k-1)$-th edge on $\rho[e]$ to build set $E[q_{c_1,c_2}]$. We will shortly prove that $E[q_{c_1,c_2}]$ is indeed a set of vertex-disjoint edges.

First, let us see two basic facts regarding $\mathcal{T}$.

LEMMA 4.4. *For every node $p \in \mathcal{T}$ with $\text{Type}[p] = 3$, let q_1, q_2 and q_{c_1,c_2}'s be all of its children defined right above. Then $\mu[q_1] + \mu[q_2] + \sum_{c_1,c_2} \mu[q_{c_1,c_2}] = \mu[p]$, and $\mu[q_2] < 400\mu[p]/\log n$.*

Proof. By definition of E_1, E_2, clearly $E[p] = E_1 \cup E_2 \cup \bigcup_{c_1,c_2} E_3[c_1, c_2]$. Thus $\mu[q_1] + \mu[q_2] + \sum_{c_1,c_2} \mu[q_{c_1,c_2}] = \mu[p]$. For the second statement, note that by definition of D, $\{e \in E[p] \setminus E_1 \mid (c_1[e], c_2[e]) = (c_1, c_2)\}$ has at most $\frac{\epsilon^2}{\log^3 n}|E[p] \setminus E_1|$ elements. Hence, ranging over all different choices of (c_1, c_2), $|E_2| \leq |C_t| \cdot (|C_t| - 1) \cdot \frac{\epsilon^2}{\log^3 n}|E[p] \setminus E_1| < 400|E[p]|/\log n$, and therefore $\mu[q_2] < 400\mu[p]/\log n$.

LEMMA 4.5. *Let p be a type-3 node with depth $< l$, and let q_1 be its unique type-1 child and let q_{c_1,c_2} be any one of its type-3 children. Then*

- *$E[q_{c_1,c_2}]$ is a set of vertex-disjoint edges.*
- *$|E[q_{c_1,c_2}]| \geq (1 - \mu[q_1]/\mu[p]) \cdot \frac{\epsilon^2 h}{2\log^3 n} \cdot |E[p]|$.*

Proof. We prove the two claims separately as following.

- For vertex-disjointness, by Lemma 2.4, any two paths $\rho[e], \rho[f], \forall e, f \in E_3[c_1, c_2]$ are vertex-disjoint or refer to the same alternating path in opposite order, since they are (c_1, c_2)-alternating paths in G_0; in the latter case, as $|\rho[e]| > 2h$, edges added to $E[q_{c_1,c_2}]$ from $\rho[e]$ and $\rho[f]$ make two vertex-disjoint edge sets.
- As for the lower bound on the size of $E[q_{c_1,c_2}]$, noting that each $e \in E_3[c_1, c_2]$ contributes exactly $h/2$ vertex-disjoint edges to $E[q_{c_1,c_2}]$, we know that $|E[q_{c_1,c_2}]| \geq h/2 \cdot \frac{\epsilon^2}{\log^3 n} \cdot |E[p] \setminus E_1| = (1 - \mu[q_1]/\mu[p]) \cdot \frac{\epsilon^2 h}{2\log^3 n} \cdot |E[p]|$.

The following establishes a connection between probability tree $\mathcal{T}$ and behavior of Algorithm 4.

LEMMA 4.6. *During the execution of Algorithm 4, at the beginning of iteration $t, \forall 1 \leq t \leq l$, for every type-3 node p of depth $t-1$, (x, y) is drawn uniformly at random on $E[p]$, with **disjoint** probability $\mu[p]$ based on random coins taken in previous iterations.*

Proof. We prove by an induction on t. For $t = 1$ the statement is trivial as $(x, y) = (u, v)$ with probability $\mu[r] = 1$. Now suppose the statement holds for some

$1 \le t < l$, and with disjoint probability $\mu[p]$, (x, y) is uniformly distributed over $E[p]$ at the beginning of iteration t, where p is a type-3 node of depth $t-1$. Then, for any type-3 child q_{c_1,c_2} of p, with disjoint probability $\mu[q_{c_1,c_2}]$, $(x, y)$is uniformly distributed on $E_3[c_1, c_2]$. Then in case this event happens, according to Algorithm 4, (z, w) is chosen from the first $h/2$ vertex-disjoint edges on the (c_1, c_2)-alternating path uniformly at random. This (c_1, c_2)-alternating path is exactly the same as $\rho[(x, y)]$ since the Algorithm 4 only modifies edges with colors from $\bigcup_{s=1}^{t-1} C_s$ in previous iterations. Hence $(x, y) = (z, w)$ is picked uniformly at random from $E[q_{c_1,c_2}]$. This concludes the induction.

COROLLARY 4.1. *The success rate of Algorithm 4 is at least* $\sum_{q\in\mathcal{T},\text{Type}[q]=1} \mu[q]$.

Proof. For any type-1 node $q_1 \in \mathcal{T}$, let p be its parent, with $\text{Type}[p] = 3$. Then, according to Lemma 4.6, with disjoint probability $\mu[p]$, variable (x, y) of Algorithm 4 follows a uniform distribution over $E[p]$. Then, with probability $\mu[q_1]$, (x, y) lies in set $E_1 = \{e \in E[p] \mid |\rho[e]| \le 2h\}$; namely, Vizing's algorithm would find an alternating path of length $\le 2h$, and thus Algorithm 4 succeeds in coloring the whole graph. Because of disjointness of events, the overall success rate of Algorithm 4 is at least $\sum_{q_1\in\mathcal{T},\text{Type}[q_1]=1} \mu[q_1]$.

By the above corollary, it suffices to prove a lower bound on $\sum_{q\in\mathcal{T},\text{Type}[q]=1} \mu[q]$. For any node $p \in \mathcal{T}$ with $\text{Type}[p] = 3$, define $\nu[p]$ to be the sum of all $\mu[q]$ where q is a descendant of p and $\text{Type}[q] = 1$. So, by definition $\nu[r] = \sum_{q\in\mathcal{T},\text{Type}[q]=1} \mu[q]$. To lower bound $\nu[r]$, we try to prove the following lemma which is more general.

LEMMA 4.7. *For any type-3 node p of depth t, $t < l$, if* $|E[p]| \ge \frac{n}{2} / \left(\frac{\epsilon^2 h}{2\log^4 n}\right)^{l-t}$, *then* $\nu[p] \ge \mu[p] \cdot (1 - 400/\log n)^{l-t}$.

Proof. We prove this by a reverse induction on t.

- Basis.

  Suppose $t = l - 1$. Let q_1 be p's type-1 child, and let q be an arbitrary type-3 child. If $\mu[q_1] < (1 - 1/\log n)\mu[p]$, then by Lemma 4.5,

$$|E[q]| \ge (1 - \mu[q_1]/\mu[p]) \cdot \frac{\epsilon^2 h}{2\log^3 n} \cdot |E[p]|$$
$$> |E[p]| \cdot \frac{\epsilon^2 h}{2\log^4 n} > n/2$$

  which is impossible as $E[q]$ is a collection of vertex-disjoint edges. Thus $\nu[p] = \mu[q_1] \ge (1 - 1/\log n)\mu[p] \ge (1 - 400/\log n)\mu[p]$.

- Induction.

  Suppose the statement works for depth $t + 1$. Consider any type-3 node p of depth t. Let q_1 be p's type-1 child, q_2 be p's type-2 child, and let $\{q_{c_1,c_2}\}$ be the set of all type-3 children. If $\mu[q_1] \ge (1 - 1/\log n)\mu[p]$, then we are done. Otherwise we assume $\mu[q_1] < (1 - 1/\log n)\mu[p]$. By Lemma 4.5, for any type-3 child q_{c_1,c_2} which has depth $t + 1$,

$$|E[q_{c_1,c_2}]| \ge (1 - \mu[q_1]/\mu[p]) \cdot \frac{\epsilon^2 h}{2\log^3 n} \cdot |E[p]|$$
$$> \frac{n}{2} / \left(\frac{\epsilon^2 h}{2\log^4 n}\right)^{l-t-1}$$

  So, by inductive hypothesis, $\nu[q_{c_1,c_2}] \ge \mu[q_{c_1,c_2}] \cdot (1 - 400/\log n)^{l-t-1}$.

  Taking a summation over all q_{c_1,c_2} and applying Lemma 4.4, we have:

$$\begin{aligned}\nu[p] &= \mu[q_1] + \sum_{c_1,c_2} \nu[q_{c_1,c_2}] \\ &\ge \mu[q_1] + (1 - 400/\log n)^{l-t-1} \cdot \sum_{c_1,c_2} \mu[q_{c_1,c_2}] \\ &\ge (1 - 400/\log n)^{l-t-1} \cdot (\mu[p] - \mu[q_2]) \\ &\ge \mu[p] \cdot (1 - 400/\log n)^{l-t}\end{aligned}$$

  which concludes the induction.

Set $h = \frac{4\log^4 n}{\epsilon^2}$ and $l = \log n$, and apply Lemma 4.7 with $t = 0$, we immediately have $\nu[r] \ge (1 - 400/\log n)^{\log n} = \Omega(1)$. Therefore, repeating Algorithm 4 for $O(\log n)$ times can blow the success rate to a high probability. As Algorithm 4 runs in $O(hl \log n) = O(\log^6 n/\epsilon^2)$ time, the overall running time becomes $O(\log^7 n/\epsilon^2)$, thus proving Theorem 1.2.

5 Extension to a changing Δ

In this section we use Theorem 1.2 as a black box to prove Theorem 1.3. The rough idea is that we guess the future Δ and maintain multiple edge colorings at the same time. Concretely speaking, let $\delta > 0$ be a constant to be determined shortly, and define a set of integers $\mathcal{D} = \{d \mid d = \lfloor(1 + 3\delta)^i\rfloor, i \ge 1\} \cap [\Omega(\log^2 n/\delta^2), n]$, and for each $d \in \mathcal{D}$ we will maintain a partial coloring of G using $(1 + 3\delta)d$ colors. For the rest of this section, we fix one parameter $d \in \mathcal{D}$ and describe how such partial coloring is maintained.

Define $V_{\text{high}} = \{u \mid \deg(u) > (1 + \delta)d\}$ and $V_{\text{low}} = \{u \mid \deg(u) \le d\}$. Each vertex has two states: *active* and *inactive*. A vertex is inactive if its most recent appearance in $V_{\text{high}} \cup V_{\text{low}}$ is in V_{high}, and otherwise it

is active. Let H be the induced subgraph of G on all active vertices, and thus the maximum degree of H is at most $(1+\delta)d$. Then we apply Theorem 1.2 on H with $\epsilon = 2\delta/(1+\delta)$ to maintain a dynamic $(1+3\delta)d$ edge coloring of H.

As the maximum degree Δ of G changes over time, we look at our data structure associated with parameter $d \in \mathcal{D}$ such that $d/(1+3\delta) < \Delta \leq d$. Then it is guaranteed that our dynamic algorithm correctly maintains a $(1+3\delta)d \leq (1+3\delta)^2\Delta = (1+O(\epsilon))\Delta$ edge coloring.

It is easy to see that H can be maintained under edge updates to G using amortized $O(1/\epsilon)$ many edge updates to H. Since Theorem 1.2 has $O(\log^7 n/\epsilon^2)$ worst-case update time, the partial coloring of H has $O(\log^7 n/\epsilon^3)$ amortized update time. Ranging over all different choices of $d \in \mathcal{D}$, the overall running time becomes $O(|\mathcal{D}|\log^7 n/\epsilon^3) = O(\log^8 n/\epsilon^4)$, which concludes the proof of Theorem 1.3.

6 Conclusion

In this paper we first propose a near-linear time algorithm for static $(1+\epsilon)\Delta$ edge coloring. Based on its techniques we devise an algorithm for $(1+\epsilon)\Delta$ edge coloring in dynamic graphs under edge insertions and deletions using poly-logarithmic update time. The major question left by our result is whether we can remove the assumption that $\Delta \geq \Omega(\log^2 n/\epsilon^2)$. Another question is whether we can further improve the current dynamic algorithm to reduce the large exponents of $\log n$ and $1/\epsilon$.

References

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

[2] Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. *Procedia Computer Science*, 108:89–98, 2017.

[3] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. SIAM, 2018.

[4] 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*, pages 2633–2652. SIAM, 2018.

[5] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-colouring bipartite multigraphs in $o(e\log d)$ time. *Combinatorica*, 21(1):5–12, 2001.

[6] Devdatt Dubhashi, David A Grable, and Alessandro Panconesi. Near-optimal, distributed edge colouring via the nibble method. *Theoretical Computer Science*, 203(2):225–252, 1998.

[7] Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné. On the decentralized dynamic graph coloring problem. *Proc. Worksh. Compl. Sys. and Self-Org. Mod*, 2007.

[8] Michael Elkin, Seth Pettie, and Hsin-Hao Su. $(2\delta$- 1)-edge-coloring is much easier than maximal matching in the distributed setting. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 355–370. Society for Industrial and Applied Mathematics, 2015.

[9] Harold N Gabow. Algorithms for edge-coloring graphs. *Technical Report TRECIS-8501, Tohoku Univ.*, 1985.

[10] Bradley Hardy, Rhyd Lewis, and Jonathan Thompson. Tackling the edge dynamic graph colouring problem with and without future adjacency information. *Journal of Heuristics*, pages 1–23, 2017.

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

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

[13] Linda Ouerfelli and Hend Bouziri. Greedy algorithms for dynamic graph coloring. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1–5. IEEE, 2011.

[14] Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.

[15] Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, and Roger Pearce. Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 347–358. IEEE, 2016.

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