# Fast and Simple Edge-Coloring Algorithms

Corwin Sinnamon\*

#### Abstract

We develop sequential algorithms for constructing edge-colorings of graphs and multigraphs efficiently and using few colors. Our primary focus is edge-coloring arbitrary simple graphs using d+1 colors, where d is the largest vertex degree in the graph. Vizing's Theorem states that every simple graph can be edge-colored using d+1 colors. Although some graphs can be edge-colored using only d colors, it is NP-hard to recognize graphs of this type [Holyer, 1981]. So using d+1 colors is a natural goal. Efficient techniques for (d+1)-edge-coloring were developed by Gabow, Nishizeki, Kariv, Leven, and Terada in 1985, and independently by Arjomandi in 1982, leading to algorithms that run in  $O(|E|\sqrt{|V|\log|V|})$  time. They have remained the fastest known algorithms for this task.

We improve the runtime to  $O(|E|\sqrt{|V|})$  with a small modification and careful analysis. We then develop a randomized version of the algorithm that is much simpler to implement and has the same asymptotic runtime, with very high probability. On the way to these results, we give a simple algorithm for (2d-1)-edge-coloring of multigraphs that runs in  $O(|E|\log d)$  time. Underlying these algorithms is a general edge-coloring strategy which may lend itself to further applications.

# 1 Introduction

An edge-coloring of a graph is an assignment of colors to edges such that any two incident edges have different colors. A k-edge-coloring is one that uses at most k colors. Edge-coloring is a standard notion in mathematics and has been well studied in many contexts; this paper is about sequential algorithms for constructing them.

Edge-coloring graphs efficiently and in few colors is a classic problem in graph algorithms, with natural applications to various scheduling tasks, e.g. [1, 9, 21, 25]. This problem has been considered in the context of simple graphs [17, 3], multigraphs [19, 22], bipartite multigraphs [2, 12, 16], and planar graphs [10, 11], to name a few. There has also been extensive work on distributed edge-coloring algorithms, e.g. [4, 18]. Some recent work has explored dynamic edge-coloring [7, 8, 13], where the coloring must be maintained as edges are inserted and deleted. Across this area of study, there is a natural tradeoff between the number colors used and the efficiency or simplicity of the algorithm. In most cases, using more colors yields simpler, faster algorithms. In general, we prefer to use as few colors as possible.<sup>1</sup>

Let G be a simple graph having n vertices and m edges, and let d be the maximum degree of any vertex in G. Clearly, at least d colors are needed to edge-color G because the edges incident to any vertex must be assigned different colors. Vizing's Theorem [24] states that G is always d+1-edge-colorable. This leaves a very small gap: G requires either d or d+1 colors. Unfortunately,

<sup>\*</sup>Department of Computer Science, Princeton University, Princeton, NJ 08540, USA. *Email:* sinncore@gmail.com

<sup>&</sup>lt;sup>1</sup>The chromatic index of G (denoted  $\chi'(G)$ ) is the smallest. number of colors in any edge-coloring of G.

<sup>&</sup>lt;sup>2</sup>Note that Vizing's theorem is only true for simple graphs; multigraphs can require up to  $\lfloor 3d/2 \rfloor$  colors [23].

distinguishing between graphs that require d colors and those that require d + 1 colors is NP-hard [20]. Since it seems infeasible to always use the optimal number of colors is infeasible, so we settle for d + 1.

It is then natural to aim for using d+1 colors. A 1985 technical report of Gabow, Nishizeki, Kariv, Leven, and Terada [17] presents an algorithm for edge-coloring simple graphs using d+1 colors.<sup>3</sup> Their algorithm runs in  $O(m\sqrt{n\log n})$  time. Although there has been significant progress for algorithms on some restricted graph classes (bipartite graphs, for example [12]), there have been no faster algorithms proposed to edge-color arbitrary simple graphs in d+1 colors. This paper improves the runtime to  $O(m\sqrt{n})$ , first by a deterministic algorithm, and then by a much simpler randomized one.

### Contributions

In this paper, we present three algorithms for edge-coloring graphs: GREEDY-EULER-COLOR, EULER-COLOR, and RANDOM-EULER-COLOR. They are founded on a general recursive strategy for edge-coloring that we lay out in detail. The strategy is a generalization of the methods used by Gabow et al. in [17], and may help in developing other edge-coloring methods.

- GREEDY-EULER-COLOR finds a (2d-1)-edge-coloring of a multigraph in  $O(m \log d)$  time. It is simple and efficient, and perhaps the most natural possible application of our strategy. Note that  $O(m \log d)$  time for (2d-1)-edge-coloring can be achieved using a known dynamic edge-coloring algorithm [7].
- EULER-COLOR is an algorithm that uses d+1 colors and runs in  $O(m\sqrt{n})$  time, improving on the result of [17]. Surprisingly, this is achieved with one subtle change to the algorithm of [17] and some additional analysis which removes a factor of  $\sqrt{\log n}$  from the runtime.
- RANDOM-EULER-COLOR is a much simpler randomized version of EULER-COLOR that uses d+1 colors and runs in  $O(m\sqrt{n})$  time with probability  $1-e^{-\operatorname{poly}(n)}$ .

We consider RANDOM-EULER-COLOR the primary contribution of this paper, due to its efficiency and simplicity. To approach the same runtime, both EULER-COLOR and the algorithm of [17] rely on a complex subroutine (COLOR-MANY in this paper and PARALLEL-COLOR in [17]). RANDOM-EULER-COLOR avoids that type of procedure using randomness, and runs just as quickly with very high probability.

We also present three coloring subroutines: Color-One, Random-Color-One, and Color-Many. They are needed by Euler-Color and Random-Euler-Color.

The paper is organized as follows. Section 2 presents the edge-coloring strategy. Sections 3, 4, and 5 introduce the three main algorithms of this paper and analyze them. The rest of the paper tackles the subroutines. Section 6 introduces the necessary definitions, Section 7 presents Color-One, Section 8 presents Random-Color-One, and Section 9 presents Color-Many.

# 2 Edge-coloring with Recursion

In this section, we present our edge-coloring strategy, setting the stage for the main algorithms of this paper. It can be approached with little preliminary discussion.

<sup>&</sup>lt;sup>3</sup>See also [3] and the discussion of that work in [17].

#### 2.1 Preliminaries

Let G = (V, E) be a undirected multigraph with n vertices, m edges, and maximum degree d (that is,  $d = \max_{v \in V} \deg(v)$ ). As shorthand, we shall write uv or vu to denote the edge  $\{u, v\}$ . We assume without loss of generality that G has no isolated vertices (vertices with no incident edges), and so  $m \ge n/2$ .

Assume we maintain a partial edge-coloring of G. An edge is *colored* if it is assigned a color in the edge-coloring and otherwise it is *uncolored*. Let  $\ell$  denote the number of uncolored edges in G. We say a color  $\alpha$  is *missing* at vertex v if no edge incident to v is colored by  $\alpha$ . Let M(v) be the set of colors missing at v.

## 2.2 Strategy

All of the algorithms in this paper employ the same strategy that we now describe. It is a generalization of a recursive method originally applied to bipartite graphs in [15] and later adapted to general graphs [17, 3]. The strategy can be applied to both simple graphs and multigraphs.

The strategy uses a recursive divide-and-conquer approach. It splits the original graph into two edge-disjoint subgraphs, recursively produces an edge-coloring of each one, and then stitches together the colorings at a reduced cost. The parameter d (the maximum degree of a vertex in the graph) is important: The subgraphs will have maximum degree roughly half that of the original graph, and consequently they can be edge-colored more efficiently and with fewer colors.

The strategy is flexible in the number of colors that can be used to edge-color the graph (the available colors). Let  $\#_c(d)$  be the number of available colors. This paper will only study  $\#_c(d) = d + 1$  and  $\#_c = 2d - 1$ , and in particular we treat  $\#_c$  as a function of d, but in principle  $\#_c$  could be any function of the graph.

The strategy works in four steps: *Partition*, *Recurse*, *Prune*, and *Repair*. The only freedom lies in the *Repair* step; the other three steps will be the same in all of our algorithms.

**Partition.** Partition the edges of the graph into two edge-disjoint subgraphs so that each subgraph has half of the edges  $(\leq \lceil m/2 \rceil)$  and the maximum degree of each subgraph is at most half the maximum degree of the original graph  $(\leq \lceil d/2 \rceil)$ . This can be done in O(m) time by the method in Section 2.3.

**Recurse.** Recurse to edge-color each subgraph using  $\#_c(\lceil d/2 \rceil)$  colors. We require that the two edge-colorings use different sets of colors; if necessary, relabel one of the edge-colorings so that no color is used in both subgraphs.

Combine the edge-colorings by taking their union. As the subgraphs are edge-disjoint and their edge-colorings use disjoint color sets, the union is an edge-coloring of the original graph.

**Prune.** At this point, the edge-coloring uses up to  $2\#_c(\lceil d/2 \rceil)$  colors, which may be more than the allowed  $\#_c(d)$  colors. (For example, if  $\#_c(d) = d + 1$ , the current coloring may use up to d+3 colors.) We must eliminate these extra colors.

Let t be number of extra colors. Choose the t least common colors and uncolor all edges with those colors. Now only  $\#_c(d)$  colors are used, and the number of uncolored edges is at most  $mt/(\#_c + t)$ .

**Repair.** To complete the coloring, we must somehow assign colors to those few uncolored edges without using more than the available colors. There is no prescribed way to do this step. We will use the subroutines mentioned at the start of this section for this purpose.

 $<sup>{}^4</sup>$ Isolated nodes are irrelevant to edge-coloring, so this is fair assumption.

Finally, if  $d \le 1$  then edge-color the graph using a single color. This serves as the base case of the recursion. The template for this strategy is presented below in pseudocode.

```
1: procedure Euler-Template(G)
    Base Case:
       If d \leq 1, color every edge by the same color and return
2:
   Partition
       Decompose G into subgraphs G_1 and G_2
3:
         {Recursively edge-color G_1 and G_2 using different sets of colors}
4:
       EULER-TEMPLATE(G_1)
5:
       EULER-TEMPLATE(G_2)
6:
         \{G \text{ is edge-colored by } \le 2\#_c(\lceil d/2 \rceil) \text{ colors}\}
7:
   Prune
       while more than \#_c colors are used do
8:
9:
           Choose the least common color \gamma
           Uncolor all edges colored by \gamma
10:
         {At most \#_c colors are used now and \ell \leq mt/(\#_c + t)}
11:
   Repair
         {Somehow color all the uncolored edges using \#_c(d) colors}
12:
13:
```

Now that the strategy is stated, the *Repair* step will be the only interesting component of our algorithms. The other steps will not change. As long as the *Repair* step stays within  $\#_c(d)$  colors and colors all the uncolored edges, the procedure must output a complete edge-coloring.

The potential of this strategy lies in the fact that only a small number of uncolored edges need to be colored at each step, and so much of the actual work of edge-coloring is done on small graphs with few edges and low maximum degree. Our Repair steps run much faster on these graphs. The efficiency of the Repair step will determine the efficiency of the whole algorithm, since the other non-recursive work done in each iteration can be executed in O(m) time.

#### 2.3 Euler Partitions

Before we proceed, let us summarize how a multigraph can be split into two edge-disjoint subgraphs as needed by the *Partition* step.

An Euler partition is a partition of the edges of a multigraph into a set of edge-disjoint tours<sup>5</sup> such that every odd-degree vertex is the endpoint of exactly one tour, and no even-degree vertex is an endpoint of a tour. Such a partition can be found greedily in O(m) time, simply by removing maximal tours of the graph until no edges remain. The edges of the graph can then be split between the subgraphs by traversing each tour and alternately assigning the edges to the two subgraphs. This yields two subgraphs, each having half of the edges (either  $\lceil m/2 \rceil$  or  $\lfloor m/2 \rfloor$ ) and maximum degree at most  $\lceil d/2 \rceil$ . The entire procedure takes O(m) time.

# 3 Greedy-Euler-Color

Our first application of the strategy is a very simple algorithm that edge-colors a multigraph in 2d-1 colors in  $O(m \log d)$  time. We state the *Repair* step; the other steps are unchanged from the

<sup>&</sup>lt;sup>5</sup>A tour is a walk that does not repeat an edge.

template.

The Repair step uses the local coloring routine GREEDY-COLOR, which takes as input an uncolored edge uv and colors it. GREEDY-COLOR simply checks each of the 2d-1 available colors to find one that is missing at both u or v. Such a color must exist by pigeonhole principle: There are at least d missing colors at each endpoint and 2d-1 colors in total. It colors uv by that color. The result is a legal partial edge-coloring, i.e. no color is reused at any vertex. This takes O(d) time in the worst case.

Greedy-Color is stated here in pseudocode. Recall that M(v) is the set of colors missing at vertex v.

```
1: procedure GREEDY-COLOR(uv)

2: for each color \alpha \in \{1, \dots, 2d-1\} do

3: if \alpha \in M(u) and \alpha \in M(v) then

4: Color uv by \alpha

5: return
```

The Repair step just applies GREEDY-COLOR to every uncolored edge. The result is a (2d-1)-edge-coloring.

**Theorem 1.** Greedy-Euler-Color edge-colors G by 2d-1 colors in  $O(m \log d)$  time.

The proof is straightforward: The *Repair* step takes O(m) time, and d is roughly halved with each recursive call. This yields a recursion tree of depth  $O(\log d)$ , where the total work done at each level in the tree is O(m). The details are left as an exercise.

# 4 Euler-Color

Assume henceforth that G is simple. EULER-COLOR builds a (d+1)-edge-coloring of G in  $O(m\sqrt{n})$  time. Here the *Repair* step will use a combination of the subroutines COLOR-ONE and COLOR-MANY. Both subroutines increase the number of colored edges in a given partial edge-coloring (without using more than d+1 colors). They are presented in Sections 7 and 9, respectively.

- Color-One colors a given uncolored edge in O(n) time.
- COLOR-MANY colors  $\Omega(\ell/d)$  of the uncolored edges in O(m) time. (Recall that  $\ell$  denotes the number of uncolored edges in the graph.)

Notice that COLOR-MANY takes amortized  $O(md/\ell)$  time for each edge it colors. Thus, as long as  $\ell = \Omega(md/n)$ , COLOR-MANY is more efficient than COLOR-ONE in terms of the cost per edge colored. This observation suggests the following *Repair* step: Use COLOR-MANY while  $\ell \geq 2md/n$ , and then switch to COLOR-ONE for the remainder.

```
1: Repair (Euler-Color)

2: {At most d+1 colors are used and \ell \leq \frac{2m}{d+3}}

3: while \ell \geq 2md/n do

4: Color-Many()

5: while \ell > 0 do

6: Choose an uncolored edge e

7: Color-One(e)
```

As discussed in the introduction, EULER-COLOR arises as a small modification to the algorithm of the same name in [17] that runs in  $O(m\sqrt{n\log n})$  time. So what is the difference between

this procedure and the original EULER-COLOR of [17]? They use a similar recursive strategy and two subroutines similar to ours: Recolor (analogous to Color-One) and Parallel-Color (analogous to Color-Many). We prefer our versions of the subroutines, but we could use Recolor and Parallel-Color instead. The only significant difference is that Parallel-Color is not integrated into the recursive strategy. Their Repair step applies Recolor to all uncolored edges. When d gets sufficiently small ( $d \le \sqrt{n/\log n}$ ), the recursive strategy is abandoned and the graph is edge-colored using only Parallel-Color. This difference increases the runtime slightly to  $O(m\sqrt{n\log n})$ . We bring Color-Many into the Repair step to further exploit the tradeoff between these two subroutines, yielding the improved time.

We now analyze the complexity of Euler-Color.

**Theorem 2.** EULER-COLOR edge-colors a simple graph using d+1 colors in  $O(m\sqrt{n})$  time.

*Proof.* The correctness of EULER-COLOR is clear from the strategy, for the *Repair* step does not terminate until the graph is completely colored and it cannot use more than d + 1 colors during that step.

Let us analyze the runtime. At the start of the *Repair* step, there are at most 2m/(d+3) uncolored edges. The *Repair* step behaves differently depending on whether d is much larger or smaller than  $\sqrt{n}$ .

If  $d < \sqrt{n}$ , then COLOR-MANY is run until the number of uncolored edges is reduced from  $\ell \le 2m/(d+3)$  to 2md/n. Let us bound the number of repetitions: A constant number of applications of COLOR-MANY decreases  $\ell$  by a factor of (1-1/d). Since  $(1-1/d)^d < 1/e$ ,

$$\ell\left(1-\frac{1}{d}\right)^{2d\ln(2\sqrt{n}/d)} < \left(\frac{2m}{d+3}\right)\left(\frac{1}{e}\right)^{2\ln(2\sqrt{n}/d)} = \left(\frac{2m}{d+3}\right)\left(\frac{d^2}{4}n\right) < \frac{2md}{n}$$

Thus it requires  $O(d \log(2\sqrt{n}/d))$  repetitions, where each repetition takes O(m) time. Afterwards it colors the remaining md/n edges in O(md) time using Color-One. Hence the Repair step takes  $O(md \log(2\sqrt{n}/d))$  time.

If  $d \ge \sqrt{n}$ , then COLOR-MANY is not run at all since  $\ell \le 2m/(d+3) \le 2md/n$ . In this case, the *Repair* step takes O(mn/d) time.

Now consider the recursion tree for EULER-COLOR(G). This is a binary tree whose root corresponds to G, and the two children of the root correspond to the subgraphs  $G_1$  and  $G_2$  constructed during the *Partition* step on G. The children of the nodes for  $G_1$  and  $G_2$  are defined recursively. A leaf corresponds to a subgraph of maximum degree 1. Identify each node with its corresponding subgraph. For a subgraph H, let  $m_H$  and  $d_H$  respectively denote the number of edges and maximum degree of H. Every subgraph contains n nodes.

Define the cost of H to be

$$cost(H) := \begin{cases} m_H n / d_H & \text{if } d_H \ge \sqrt{n} \\ m_H d_H \log(2\sqrt{n} / d_H) & \text{if } d_H < \sqrt{n} \end{cases}$$

This cost captures (up to a constant) all the time spent on the subgraph H, according to our earlier analysis. Thus, the total time for Euler-Color is proportional to the sum of costs of all subgraphs in the recursion tree.

Consider the sum of costs over high-degree subgraphs: the subgraphs H with  $d_H \ge \sqrt{n}$ . (We can assume here that  $d \ge \sqrt{n}$ , for otherwise there would be no high-degree subgraphs.) These subgraphs have depth at most  $\log(2d/\sqrt{n})$ , since the vertex degrees roughly halve between a subgraph and its

child. Moreover, the subgraphs at a given depth partition the edges of G, so the sum of their edge counts is exactly m. Hence, the total cost of all high-degree subgraphs is at most

$$\sum_{t=0}^{\log(2d/\sqrt{n})} \frac{mn}{d} = \frac{mn}{d} \log(2d/\sqrt{n}) \le \frac{mn}{\sqrt{n}} \log(2\sqrt{n}/\sqrt{n}) = m\sqrt{n}$$

Now consider the total cost of the remaining low-degree subgraphs, which have  $d_H < \sqrt{n}$ . Fix such a subgraph H, and let  $H_1$  and  $H_2$  be its children. Observe that

$$cost(H_1), cost(H_2) \le \left(\frac{m_H + 1}{2}\right) \left(\frac{d_H + 1}{2}\right) \log(4\sqrt{n}/d) \approx \frac{m_H d_H}{4} \log(2\sqrt{n}/d) \approx cost(H)/4$$

That is, the cost shrinks by a factor of 4, roughly, from a parent to its child. Even though it does not shrink by a factor of 4 exactly, it is not hard to show that it shrinks by at least a factor of 3, for sufficiently large  $d_H$ . Hence, cost decreases geometrically with depth, shrinking by a factor of at least 3 at each level (except possibly near the bottom of the tree, vertex degrees are less than some constant).

Why does this matter? Each subgraph H has 2 children, 4 grandchildren, 8 great-grandchildren, etc. So H has  $2^t$  descendants at a depth t levels lower than H. But the cost of each descendant is at most  $\frac{1}{3^t} \cos(H)$ . Hence the *combined* cost of the descendants of H decreases as a geometric series:  $\sum_{t=0}^{\infty} \left(\frac{2}{3}\right)^t \cos(H) = O(\cos t H)$ .

Therefore, the cost of all low-degree subgraphs is only a constant factor larger than the cost of the "maximal" low-degree subgraphs (those whose parents are high-degree).

In the case that  $d \geq \sqrt{n}$ , there are multiple maximal subgraphs, and partition the edges of G. Moreover, each maximal subgraph H has  $d_H \in [\sqrt{n}/2, \sqrt{n})$ . Hence, their combined cost is at most  $m\sqrt{n}\log(4\sqrt{n}/\sqrt{n}) = O(m\sqrt{n})$ .

If  $d < \sqrt{n}$ , G is the only such maximal subgraph, and so the total cost is  $O(\cos(G)) = O(md\log(2\sqrt{n}/d))$ . Observe that  $md\log(2\sqrt{n}/d) \le m\sqrt{n}$  for all  $d < \sqrt{n}$ . Hence the total cost is  $O(m\sqrt{n})$ .

The trouble with EULER-COLOR is that COLOR-MANY is complicated, much more so than the rest of the algorithm. Thankfully we can simplify enormously by injecting a little randomness.

# 5 Random-Euler-Color

We now present RANDOM-EULER-COLOR, a randomized version of EULER-COLOR. It also uses d+1 colors and runs in  $O(m\sqrt{n})$  time with high probability.

We use a subroutine RANDOM-COLOR-ONE, a randomized extension to COLOR-ONE, that is presented in Section 8.7

• RANDOM-COLOR-ONE colors an uncolored edge in  $O(md/\ell)$  expected time, or O(n) time in the worst case.

 $<sup>^6</sup>$ A constant number of levels near the bottom of the tree (for which the d value is constant) may not follow this geometric trend, but their total cost is negligible.

<sup>&</sup>lt;sup>7</sup>RANDOM-COLOR-ONE essentially applies COLOR-ONE to an uncolored edge *chosen uniformly at random*. It also chooses one other variable (internal to COLOR-ONE) uniformly at random. These small changes are enough to improve the expected run time.

Notice that the expected time matches the  $O(md/\ell)$  amortized time-per-edge-colored that Color-Many achieves. It also has the same O(n) worst-case time bound of Color-One, meaning that Random-Color-One is an effective drop-in replacement for *both* Color-One and Color-Many.

The Repair step just applies RANDOM-COLOR-ONE until the edge-coloring is complete.

- 1: **Repair** (Random-Euler-Color)
- 2: {At most d+1 colors are used and  $\ell \leq \frac{2m}{d}$ }
- 3: while  $\ell > 0$  do
- 4: Random-Color-One()

Now we analyze the expected runtime — but the hard work has already been done! The Repair step of Euler-Color used a combination of Color-One and Color-Many to color the O(m/d) uncolored edges. We notice that Random-Color-One performs as well in expectation as either subroutine. Hence, the Repair step of Random-Euler-Color can be no slower than that of Euler-Color. Indeed, direct analysis of the Repair step leads to the same cost function that appeared in our analysis of Euler-Color:

$$\sum_{\ell=1}^{2m/d} \min\left(n, \frac{md}{\ell}\right) = \begin{cases} O(mn/d) & \text{if } d \ge \sqrt{n} \\ O(md\log(2\sqrt{n}/d)) & \text{if } d < \sqrt{n} \end{cases}$$

By Theorem 2, we obtain a tight bound on the expected runtime of RANDOM-EULER-COLOR.

**Lemma 3.** Random-Euler-Color edge-colors a simple graph by d+1 colors in  $O(m\sqrt{n})$  expected time.

Now that we have established the expected time, we can go further. In fact,  $O(m\sqrt{n})$  time is achieved except with *exponentially* small probability, and we can show this much stronger claim with a little work.

**Theorem 4.** Random-Euler-Color runs in  $O(m\sqrt{n})$  time with probability  $1 - e^{-\sqrt{m}}$ .

*Proof.* Let T(G) be the combined runtime of all calls to RANDOM-COLOR-ONE during an execution of RANDOM-EULER-COLOR on G. All other operations in RANDOM-EULER-COLOR are deterministic and run in  $o(m\sqrt{n})$  time, so it suffices to study T(G).

RANDOM-COLOR-ONE is applied many times during RANDOM-EULER-COLOR to many partially edge-colored subgraphs of G. Call them  $G_1, \ldots, G_k$ , where  $G_i$  has  $m_i$  edges, maximum degree  $d_i$ , and  $\ell_i$  uncolored edges. Note that these are not random variables. The graphs  $G_1, \ldots, G_k$  are determined uniquely by the input graph because the *Partition* step is deterministic and depends only on the structure of the graph. Although the partial edge-colorings are randomly altered by RANDOM-COLOR-ONE, the values of  $m_i$ ,  $d_i$ , and  $\ell_i$  are predictable.

Let  $R_i$  be a random variable for the runtime of RANDOM-COLOR-ONE on  $G_i$ . Since the  $G_i$  are deterministically chosen, the  $R_i$  are independent. Then  $T(G) = \sum_{i=1}^k R_i$ . Lemma 3 states that  $\mathbb{E}[T(G)] = \mathbb{E}\left[\sum_{i=1}^k R_k\right] \leq Cm\sqrt{n}$ , for a sufficiently large constant C.

Recall that RANDOM-COLOR-ONE takes O(n) time in the worst case, and so  $R_i \leq Cn$  (we reuse the constant C here, without loss of generality). Hence  $\operatorname{Var}[R_i] \leq \mathbb{E}[R_i^2] \leq Cn \mathbb{E}[R_i]$  for  $i = 1, \ldots, k$ . By independence, we have

$$\operatorname{Var}[T(G)] \le Cn \operatorname{\mathbb{E}}[T(G)] \le C^2 mn^{3/2}$$

We now use the following bound of Bernstein, proved in [6] and communicated in [5].

**Theorem** (Bernstein [6]). Let  $X_1, \ldots, X_n$  be independent random variables, let  $\mathbf{X} = \sum_{i=1}^n X_i$ , and let  $\sigma^2 = \operatorname{Var}[\mathbf{X}] = \sum_{i=1}^n \operatorname{Var}[X_i]$ . Suppose that  $|X_i - \mathbb{E}[X_i]| \leq M$  for all i. Then, for t > 0,

$$\mathbb{P}\left[\mathbf{X} \ge \mathbb{E}[\mathbf{X}] + t\sigma\right] \le \exp\left(-\frac{\frac{1}{2}t^2}{1 + \frac{1}{3}\frac{M}{\sigma}t}\right)$$

Apply this theorem to T(G). Here M = Cn and  $\sigma^2 = \text{Var}[T(G)] \leq C^2 m n^{3/2}$ , and we set  $t = 3m^{1/2}/n^{1/4}$ . We have  $t\sigma \leq 3Cm\sqrt{n}$ . Hence,

$$\mathbb{P}\left[T(G) \ge \mathbb{E}[T(G)] + 3Cm\sqrt{n}\right] \le \exp\left(-\frac{\frac{9}{2}m/\sqrt{n}}{1 + \frac{1}{3}\frac{Cn}{Cm^{1/2}n^{3/4}}\frac{3m^{1/2}}{n^{1/4}}}\right) = \exp\left(-\frac{\frac{9}{2}m/\sqrt{n}}{2}\right) \le e^{-\sqrt{m}}$$

The last inequality holds because  $m \ge n/2$ , by the assumption that there are no isolated nodes in any subgraph. Thus, the runtime of RANDOM-EULER-COLOR is  $O(m\sqrt{n})$  except with probability  $e^{-\sqrt{m}}$ .

In light of its strongly concentrated runtime and the simplicity of Color-One compared to Color-Many, Random-Euler-Color is preferable to Euler-Color in almost any context.

This concludes our study of the three major algorithms of this paper. It remains to present the subroutines used by EULER-COLOR and RANDOM-EULER-COLOR.

# 6 Colors, Fans, and Paths

The rest of the paper is devoted to presenting the subroutines Color-One, Color-Many, and Random-Color-One that are used in Sections 4 and 5. This section provides some terminology and tools that we will need to describe them, including the data structures underlying our implementations.

#### 6.1 Colors

Since the remainder of this paper is concerned with edge-coloring in d+1 colors, we shall assume the available colors are  $1, 2, \ldots, d+1$ . We also restrict ourselves to simple graphs.

Let G = (V, E) be a simple undirected graph with maximum degree d. Let  $c: E \to [d+1]$  be a partial edge-coloring of G. An edge e is said to be colored if c(e) is defined and uncolored otherwise. Recall that a color  $\alpha$  is said to be missing at vertex v if no edge incident to v is colored by  $\alpha$ . Define  $M(v) = \{\alpha \in [d+1] \mid \alpha \text{ is missing at } v\}$  and  $\overline{M}(v) = [d+1] \setminus M(v)$ . Observe that M(v) is always nonempty as v has at most d neighbours and there are d+1 colors to choose from.

#### **6.2** Fans

Our subroutines will use a construction called a c-fan (sometimes called a Vizing fan), illustrated in Figure 1.

**Definition 5.** A c-fan F is a sequence  $(\alpha, v, x_0, x_1, \dots, x_k)$  such that

- $\alpha$  is a color in M(v),
- $x_1, \ldots, x_k$  are distinct neighbours of v,
- $vx_0$  is uncolored,

- $vx_i$  is colored for i = 1, ..., k, and
- $c(vx_i)$  is missing at  $x_{i-1}$  for i = 1, ..., k.

Vertex v is called the center of F, and F is said to be centered at v. The other vertices  $(x_0, \ldots, x_k)$  are called the leaves of F.

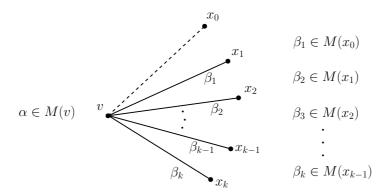


Figure 1: A c-fan  $(\alpha, v, x_0, x_1, \dots, x_k)$ . Here  $c(vx_i) = \beta_i$  and  $\beta_i$  is always missing at  $x_{i-1}$ .

The useful property of a c-fan is that we may "rotate" the colors of the fan without making the edge-coloring invalid. Let  $F = (\alpha, v, x_0, \dots, x_k)$  be a c-fan with  $\beta_i = c(vx_i)$  for  $i = 1, \dots, k$ , and let  $0 \le j \le k$ . To shift F from  $x_j$  means to set  $c(vx_{i-1}) = \beta_i$  for  $i = 1, \dots, j$ , and to make  $vx_j$  uncolored. Since  $\beta_i$  is required to be missing at  $x_{i-1}$ , the function c is still a partial edge-coloring after a shift. Note that M(v) is unchanged and that  $F' = (\alpha, v, x_j, x_{j+1}, \dots, x_k)$  is a c-fan after the shift. Shifting a c-fan is shown in Figure 2.

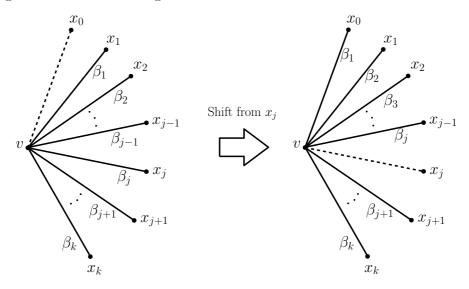


Figure 2: A c-fan (left) and the resulting edge-coloring after shifting from  $x_j$  (right). Solid lines are colored and dashed lines uncolored. A shift always preserves the legality of the edge-coloring.

Let us extend the definition of a c-fan by adding a parameter  $\beta$ . Intuitively, a c-fan is *primed* by  $\beta$  once it is complete and does not need to be made larger. A primed c-fan is exactly what we need to grow the edge-coloring, as we will see in the next sections.

**Definition 6.** A primed c-fan is a c-fan  $F = (\alpha, v, x_0, \dots, x_k)$  with an extra parameter  $\beta$ . The color  $\beta$  must satisfy one of the following conditions:

- 1.  $\beta \in M(x_k)$  and  $\beta \in M(v)$ , or
- 2.  $\beta \in M(x_k)$  and there is a leaf  $x_j$  such that  $c(vx_j) = \beta$ .

Setting  $\beta$  is called priming F, and we say F is primed by  $\beta$ .

# 6.3 Alternating Paths

We require one more tool to describe our algorithms. For any two colors  $\alpha$  and  $\beta$ , restricting G to the edges colored by  $\alpha$  or  $\beta$  yields a subgraph of G in which every component is either a path or an even-length cycle. We call each component an  $\alpha\beta$ -path or an  $\alpha\beta$ -cycle, as appropriate. We shall not require that  $\alpha$  and  $\beta$  be distinct, and an  $\alpha\beta$ -path may consist of a single vertex. An  $\alpha\beta$ -path may also be called an alternating path when  $\alpha$  and  $\beta$  are not specified.

To *flip* an alternating path means to interchange the colors of its edges. Any alternating path can be flipped in a partially edge-colored graph and the resulting assignment of colors will still be a partial edge-coloring.

Using the data structures in the next section, an alternating path can be flipped in time proportional to its length.

#### 6.4 Data Structures

Our algorithms use the following elementary data structures. Here we diverge from [17] by introducing a dictionary data structure that simplifies our subroutines. See Appendix A for further discussion of this structure.

- We represent G by storing for each vertex a list of its incident edges. Each edge has a color field to track its color, if any.
- We keep a data structure  $\mu(v)$  that stores colors missing at each vertex v. However,  $\mu(v)$  will not store all the colors in M(v), as that could require too much time and space to initialize. Instead  $\mu(v)$  contains only the colors in  $M(v) \cap [\deg(v) + 1]$ . There must always be some color in  $\mu(v)$  because  $M(v) \cap [\deg(v) + 1]$  is always nonempty.
  - We implement  $\mu(v)$  as a doubly-linked list containing those colors in  $M(v) \cap [\deg(v) + 1]$  together with an array of length  $\deg(v) + 1$ . The array contains pointers to the  $\deg(v) + 1$  nodes that can be in the list. We can maintain  $\mu(v)$  with constant overhead: Every time an edge incident to v is colored by  $\gamma \in [\deg(v) + 1]$ , find the list node for  $\gamma$  (using the array) and remove it from the list. Every time an edge incident to v with a color in  $[\deg(v) + 1]$  is uncolored, append the node for that color to the linked list. We can initialize  $\mu(v)$  in  $O(\deg(v))$  time.
- We also use a dictionary data structure  $\mathcal{D}$  that maps a vertex-color pair  $(v, \gamma)$  to the edge incident to v colored by  $\gamma$ , if there is such an edge. It supports three operations that each take constant time: Search, Insert, and Delete.
  - $\mathcal{D}$  can be initialized in  $O(\sqrt{mnd})$  time, which is no more than  $O(m\sqrt{n})$ . Our implementation of  $\mathcal{D}$  is unusual because it must achieve this low initialization cost, but it operates by elementary methods. In Appendix A, we describe the implementation of  $\mathcal{D}$  and how to integrate  $\mathcal{D}$  into the recursive algorithms of Section 2. The latter topic merits discussion because  $\mathcal{D}$  can only be initialized once, yet it must work for all the subgraphs in the recursion.

A data structure like  $\mathcal{D}$  was not present in [17]. It allows us to speed up or simplify all of our subroutines, and it is not known how to make a procedure like RANDOM-COLOR-ONE function without such a data structure.

In summary, our data structures permit the following operations in constant time for any vertex v and color  $\gamma$ : check whether  $\gamma \in M(v)$ ; get a color in M(v); color an edge; uncolor an edge; find the edge incident to v colored by  $\gamma$ . As a corollary, we are able to shift a c-fan in time proportional to its size and flip an alternating path in time proportional to its length.

# 7 Color-One

Our basic edge-coloring subroutine COLOR-ONE is analogous to RECOLOR in [17] and derived from Vizing's original work [24].

COLOR-ONE first chooses an uncolored edge  $vx_0$  and a color  $\alpha \in M(v)$ . It calls a procedure Make-Primed-Fan, which builds a fan  $F = (\alpha, v, x_0, x_1, \dots, x_k)$  primed by a color  $\beta \in M(x_k)$ . It then *activates* the fan by calling Activate-C-Fan, which uses the fan F to color  $vx_0$ .

```
    procedure Color-One()
    Choose an uncolored edge vx<sub>0</sub>
    Choose any α ∈ M(v)
    F ← Make-Primed-Fan(v, x<sub>0</sub>, α)
    Activate-C-Fan(F)
    We describe Make-Primed-Fan and Activate-C-Fan in turn.<sup>8</sup>
```

#### Make-Primed-Fan

MAKE-PRIMED-FAN creates a primed c-fan around  $vx_0$ , as follows. Initialize a fan  $F = (\alpha, v, x_0)$ . We add leaves to F until we find an opportunity to prime F. Leaves are added greedily. Pick a color  $\beta$  missing at  $x_k$ , the last leaf of F. If  $\beta$  is missing at v, then we can prime F by  $\beta$ ; this satisfies the first way for F to be primed. Otherwise, there is a vertex  $x_{k+1}$  with  $vx_{k+1}$  colored by  $\beta$ . If  $x_{k+1}$  is already a leaf of F, then we can again primes F by  $\beta$ ; this satisfies the second way for F to be primed. If  $x_{k+1}$  does not already appear in F, then appended to F as the last leaf. Return F as soon as it is primed.

```
1: procedure Make-Primed-Fan(v, x_0, \alpha)
Require: vx_0 uncolored
Require: \alpha \in M(v)
         F \leftarrow (\alpha, v, x_0)
        k \leftarrow 0
 3:
         while F is not primed do
 4:
             Pick any \beta \in M(x_k)
 5:
             if \beta \in M(v) then
 6:
                 Prime F by \beta
 7:
             else \{\beta \notin M(v)\}
 8:
                 Find neighbour x_{k+1} such that c(vx_{k+1}) = \beta
 9:
                 if x_{k+1} \in \{x_1, ..., x_k\} then
10:
                     Prime F by \beta
11:
```

<sup>&</sup>lt;sup>8</sup>In most presentations of this type of coloring procedure, the construction of the c-fan and the activation of the fan are not treated as separate components. This bulky presentation will help us when we state RANDOM-COLOR-ONE and COLOR-MANY.

```
12: else \{x_{k+1} \text{ is not already in } F\}
13: Append x_{k+1} to F
14: k \leftarrow k+1
15: return F
```

**Lemma 7.** Make-Primed-Fan returns a primed c-fan in  $O(\deg(v))$  time.

*Proof.* By construction, F is a c-fan at all times during Make-Primed-Fan. Each iteration of the loop adds a new leaf to F or primes F, and no leaf can be added twice; thus the loop repeats at most  $\deg(v)$  times before F is primed and returned.

Each operation within the loop takes constant time — this is clear for every line except line 10, where we check whether  $x_{k+1}$  appears already as a leaf of F. We can make this operation take constant time by marking each leaf when it is added to F and unmarking them all at the end. Hence the procedure takes  $O(\deg(v))$  time.

#### Activate-c-Fan

ACTIVATE-C-FAN accepts a c-fan  $F = (\alpha, v, x_0, x_1, \dots, x_k)$  primed by  $\beta$  and uses F to extend the coloring. By Definition 6, there are two ways for F to be primed by  $\beta$ : Either  $\beta \in M(v)$  or there is a leaf  $x_j$  with  $c(vx_j) = \beta$  (recall that  $\beta \in M(x_k)$  in both cases). ACTIVATE-C-FAN handles them differently.

If  $\beta \in M(v)$ , then shift F from  $x_k$  (so that  $vx_k$  becomes uncolored) and color  $vx_k$  by  $\beta$ . This extends the coloring by one edge, and we return.

Otherwise,  $\beta$  is not missing at v, but there is some leaf  $x_j$  where  $vx_j$  is colored by  $\beta$ . Observe that  $\beta$  is missing at  $x_k$  and at  $x_{j-1}$ , by construction of the c-fan.

Consider the  $\alpha\beta$ -path P beginning at v. Recall that  $\alpha$  is missing at v and  $vx_j$  is colored by  $\beta$ . Hence, v is an endpoint of P, with  $vx_j$  being the first edge of P. Path P ends at some other vertex w (note that w is missing exactly one of  $\alpha$  or  $\beta$ ). Flip P (i.e. interchange  $\alpha$  and  $\beta$  on its edges). After the flip,  $vx_j$  is colored by  $\alpha$ , and  $\beta$  is missing at v. The flip also affects w: If  $\alpha$  was missing at w, now  $\beta$  is missing, and vice versa. The set of missing colors does not change for any vertices besides v and w.

We now act in two cases depending on w. The cases are illustrated in Figure 3.

- Case I: If  $w \neq x_{j-1}$ , then  $\beta$  is still missing at  $x_{j-1}$ . Observe that  $F' = (\beta, v, x_0, \dots, x_{j-1})$  is a c-fan (this holds even if w is a leaf of F'). Shift F' from  $x_{j-1}$  and color  $vx_{j-1}$  by  $\beta$ .
- Case II: If  $w = x_{j-1}$ , then  $\beta$  is no longer missing at  $x_{j-1}$  (now  $\alpha$  is missing instead). Since  $vx_j$  is colored by  $\alpha$  and  $\beta$  is missing at v, the sequence  $F' = (\beta, v, x_0, x_1, \dots, x_k)$  is a c-fan. Moreover,  $\beta$  must still be missing at  $x_k$  because  $w \neq x_k$ . Shift F' from  $x_k$  and color  $vx_k$  by  $\beta$ .

In either case, the procedure succeeds in extending the edge-coloring by  $vx_0$  without uncoloring any other vertices.

ACTIVATE-C-FAN is given in pseudocode below.

```
1: procedure Activate-c-Fan(F)

Require: F = (\alpha, v, x_0, \dots, x_k) is a c-fan primed by \beta

2: if \beta \in M(v) then

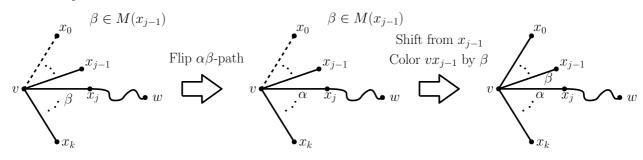
3: \{\beta \in M(v) \text{ and } \beta \in M(x_k)\}

4: Shift F from x_k

5: Color vx_k by \beta
```

```
else \{\beta \notin M(v)\}
 6:
             Let x_i be the leaf of F with c(vx_i) = \beta
 7:
               \{\beta \in M(x_{j-1}) \text{ and } \beta \in M(x_k)\}
 8:
             Flip the \alpha\beta-path P beginning at v
 9:
             Let w be the other endpoint of P
10:
             if w \neq x_{i-1} then
11:
                    \{\beta \in M(x_{j-1})\}
12:
13:
                  Shift F from x_{i-1}
                  Color vx_{i-1} by \beta
14:
15:
             else
                    \{\alpha \in M(x_{j-1})\}
16:
                  Shift F from x_k
17:
                  Color vx_k by \beta
18:
```

Case I:  $w \neq x_{j-1}$ 



Case II:  $w = x_{i-1}$ 

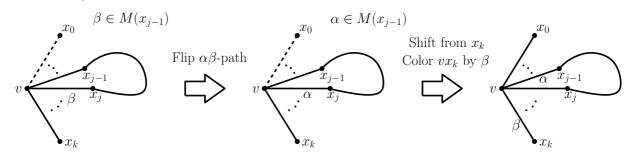


Figure 3: The two cases of flipping the  $\alpha\beta$ -path in ACTIVATE-C-FAN.

For convenience, let us assume going forward that ACTIVATE-C-FAN always flips one  $\alpha\beta$ -path with an endpoint at v. This is indeed the case if ACTIVATE-C-FAN reaches line 9. If it instead executes lines 3—5, we can just pretend that the  $\alpha\beta$ -path beginning at v is flipped; it does no harm to assume this because the path has length 0.

**Lemma 8.** ACTIVATE-C-FAN colors one edge in  $O(\deg(v) + \operatorname{length}(P))$  time, where P is the length of the flipped path.

*Proof.* Case analysis of ACTIVATE-C-FAN shows that it outputs a legal edge-coloring. Observe that no previously-colored edges are uncolored by ACTIVATE-C-FAN, and the first edge of the fan becomes colored. Hence it increases the number of colored edges by one.

The runtime is clear, as ACTIVATE-C-FAN takes  $O(\deg(v))$  time for all operations except for flipping the path, which takes  $O(\operatorname{length}(P))$  time.

Lemmas 7 and 8 complete the analysis of Color-One. Theorem 9 shows that Color-One matches the behaviour promised in Section 4.

**Theorem 9.** Color-One colors an uncolored edge  $vx_0$  in  $O(\deg(v) + \operatorname{length}(P))$  time in the worst case.

## 8 Random-Color-One

While Color-One can take  $\Theta(n)$  time when the flipped path is very long, we might expect better behaviour "on average". Indeed, it cannot happen that every alternating path induced by an edge-coloring is very long. It turns out that randomization helps a great deal in expectation.

The intuition is this: Let Color-One choose  $vx_0$  at random from the uncolored edges. Color-One can only flip an alternating path if v is an endpoint of the path. Since there are many different edges  $vx_0$  to choose from, any particular alternating path is unlikely to be chosen.

This intuition is not far off, but it glosses over an important fact: If a vertex v is incident to many uncolored edges, it is much more likely to be chosen for Color-One. If v happens to be the endpoint of a very long path, Color-One may be slow in expectation. To compensate for this, we also choose  $\alpha$  at random. The flipped path must use  $\alpha$  as one of its colors, so more choices for  $\alpha$  translates to more options for the flipped path. Moreover, if v is incident to many uncolored edges, it must also be missing many colors.

In summary, run Color-One with  $vx_0$  and  $\alpha$  chosen randomly. We call this randomized procedure Random-Color-One.

- 1: procedure RANDOM-COLOR-ONE()
- 2: Choose an uncolored edge  $vx_0$  uniformly at random
- 3: Choose  $\alpha \in M(v)$  uniformly at random
- 4:  $F \leftarrow \text{Make-Primed-Fan}(v, x_0, \alpha)$
- 5: ACTIVATE-C-FAN(F)

**Lemma 10.** Random-Color-One colors an uncolored edge. It takes  $O(md/\ell)$  time in expectation and O(n) time in the worst case.

*Proof.* The correctness of RANDOM-COLOR-ONE follows from Lemmas 7 and 8, as does the worst-case O(n) runtime. It remains to show the expected time.

RANDOM-COLOR-ONE chooses v,  $x_0$ , and  $\alpha$  by a random process. Treat these as random variables and let P be a random variable representing the flipped path. Since RANDOM-COLOR-ONE runs in O(d + length(P)) time, it is sufficient to show that  $\mathbb{E}[\text{length}(P)] = O(md/\ell)$ .

Let Q be an alternating path in the graph with endpoints x and y. Since they are endpoints, x and y must each be missing exactly one of the colors that appear on Q. Say  $\gamma_x \in M(x)$  and  $\gamma_y \in M(y)$  are both colors used in Q. In order for P to take the value Q, we need either v = x and  $\alpha = \gamma_x$  or v = y and  $\alpha = \gamma_y$ .

Since  $vx_0$  is chosen uniformly at random from the uncolored edges, we have

$$\mathbb{P}[v=x] = \frac{\deg(x) - |\overline{M}(x)|}{2\ell} \le \frac{|M(x)|}{2\ell}$$

and similarly  $\mathbb{P}[v=y] \leq |M(y)|/2\ell$ . Since  $\alpha$  is chosen uniformly at random from M(v),

$$\begin{split} \mathbb{P}[P = Q] &\leq \mathbb{P}[v = x \text{ and } \alpha = \gamma_x] + \mathbb{P}[v = y \text{ and } \alpha = \gamma_y] \\ &= \mathbb{P}[v = x] \, \mathbb{P}[\alpha = \gamma_x \mid v = x] + \mathbb{P}[v = y] \, \mathbb{P}[\alpha = \gamma_y \mid v = y] \\ &\leq \frac{|M(x)|}{2\ell} \cdot \frac{1}{|M(x)|} + \frac{|M(y)|}{2\ell} \cdot \frac{1}{|M(y)|} \\ &= \frac{1}{\ell} \end{split}$$

Now, letting Q range over all alternating paths in G, we have

$$\mathbb{E}[\operatorname{length}(P)] = \sum_{Q} \operatorname{length}(Q) \cdot \mathbb{P}[P = Q] \le \sum_{Q} \frac{\operatorname{length}(Q)}{\ell}$$

Observe that every colored edge in G contributes to at most d+1 different alternating paths in G. Hence  $\sum_{Q} \operatorname{length}(Q) \leq m(d+1)$ , and so  $\mathbb{E}[\operatorname{length}(P)] \leq m(d+1)/\ell$ .

Thus, with this trivial modification to Color-One, the runtime is much improved. It is especially significant because it does as well (in expectation) as the better of Color-One and Color-Many in terms of the time per edge colored, yet it is no more difficult to implement than Color-One. As we showed via Random-Euler-Color in Section 5, Random-Color-One is all we need to edge-color a graph in  $O(m\sqrt{n})$  time with high probability.

# 9 Color-Many

This section presents Color-Many, a subroutine that colors  $\Omega(\ell/d)$  uncolored edges in a graph in O(m) time. In other words, it colors edges at an average cost of  $O(md/\ell)$  time per edge. In that sense, Color-Many serves as a deterministic replacement for Random-Color-One. We used Color-Many as a subroutine in Euler-Color to achieve a runtime of  $O(m\sqrt{n})$ .

Color-Many is our version of Parallel-Color, a procedure presented in [17]. As far as our results are concerned, the differences between Color-Many and Parallel-Color are unimportant—they have the same asymptotic efficiency per edge that they color. Color-Many uses the essential ideas and mechanisms of Parallel-Color, and indeed they are similar in many ways. We offer a more complete and thorough analysis of the procedure and we introduce some new ideas in Color-Many to simplify the presentation and implementation. Although Parallel-Color could be used in place of Color-Many, we believe that Color-Many is preferable.

Despite these improvements, Color-Many is technical and requires some care. Before we begin, we need some additional machinery.

#### Coloring with u-Fans

We need a second kind of fan called a *u-fan*. Like a c-fan, a u-fan can be used to color an uncolored edge by a procedure we call ACTIVATE-U-FAN.

**Definition 11.** A u-fan F is a sequence  $(\alpha, v, x_1, \dots, x_k)$  where  $k \geq 2$ , such that

- $x_1, \ldots, x_k$  are distinct vertices that are all incident to v,
- $vx_i$  is uncolored for i = 1, ...k, and

•  $\alpha$  is a color that is missing at every  $x_i$  and not missing at v; that is,

$$\alpha \in \overline{M}(v) \cap \bigcap_{i=1}^{k} M(x_i).$$

Vertex v is called the center of F, and F is said to be centered at v. The vertices  $x_1, \ldots, x_k$  are called the leaves of F.

Note that we require a u-fan to have at least two leaves. We need this property to reliably color an uncolored edge using a u-fan. If a u-fan is reduced to have less than two leaves, we call it degenerate.

ACTIVATE-U-FAN takes a u-fan  $F = (\alpha, v, x_1, \dots, x_k)$  and a color  $\beta$  that is missing at v. It flips the  $\alpha\beta$ -path beginning at v so that  $\alpha$  is missing at v. If the flipped path ends at one of the leaves, then  $\alpha$  is no longer missing there, so that leaf is removed from F. A leaf x of F is chosen and vx is colored by  $\alpha$ . Then x is removed from F so that the resulting fan is still a u-fan (possibly a degenerate one).

1: **procedure** ACTIVATE-U-FAN $(F, \beta)$ 

**Require:**  $F = (\alpha, v, x_1, \dots, x_k)$  is a u-fan

Require:  $\beta \in M(v)$ 

- 2: Flip the  $\alpha\beta$ -path P beginning at v
- 3: Let w be the endpoint of P
- 4: **if** w is a leaf of F then
- 5: Remove w from F
- 6: Pick any leaf x of F
- 7: Color vx by  $\alpha$
- 8: Remove x from F

ACTIVATE-U-FAN takes time proportional to the length of the flipped path. It always colors an edge and F remains a (possibly degenerate) u-fan. Since the colored edge is missing  $\alpha$  at both endpoints, the partial edge-coloring of the graph remains legal.

### Outline of Color-Many

COLOR-MANY is based on a set of disjoint fans that use the same color  $\alpha$  as their first parameter. We refer to such a set as an  $\alpha$ -collection. Formally, for a color  $\alpha \in [d+1]$ :

**Definition 12.** An  $\alpha$ -collection is a set  $\mathcal{C}_{\alpha}$  of vertex-disjoint fans such that every fan  $F \in \mathcal{C}_{\alpha}$  is either a primed c-fan  $(\alpha, v, x_0, x_1, \dots, x_k)$  or a u-fan  $(\alpha, v, x_1, \dots, x_k)$ . Let  $V(\mathcal{C}_{\alpha})$  denote the set of vertices in all the fans of  $\mathcal{C}_{\alpha}$ .

COLOR-MANY builds an  $\alpha$ -collection  $\mathcal{C}_{\alpha}$  around a set of uncolored edges where each edge has one endpoint missing  $\alpha$ . Then it uses  $\mathcal{C}_{\alpha}$  to color a large fraction of those edges. It proceeds in three steps, each requiring O(m) time, that work broadly as follows.

#### Step 1: Choose $\alpha$

Say a vertex is *incomplete* if it is incident to an uncolored edge. Let I be the set of incomplete vertices. For any color  $\beta$ , let  $I_{\beta} = \{v \in I \mid \beta \in M(v)\}$ . Choose  $\alpha$  to maximize  $|I_{\alpha}|$ .

#### Step 2: Build an $\alpha$ -collection

Build an  $\alpha$ -collection  $\mathcal{C}_{\alpha}$  using the vertices of  $I_{\alpha}$ . The goal is for many vertices of  $I_{\alpha}$  to end up in  $V(\mathcal{C}_{\alpha})$  as the centers of primed c-fans or the leaves of u-fans. Color-Many does this by a procedure called Build-Collection.

### Step 3: Activate the $\alpha$ -collection

Activate each fan in  $\mathcal{C}_{\alpha}$  using ACTIVATE-C-FAN or ACTIVATE-U-FAN, as appropriate. After an activation, find any other fans in the collection that have been damaged and either repair them or discard them so that  $\mathcal{C}_{\alpha}$  remains an  $\alpha$ -collection. Color-Many does this by a procedure called ACTIVATE-COLLECTION.

Note that edges may be colored in both step 2 and step 3.

- 1: **procedure** Color-Many()
- 2: Choose  $\alpha$  that maximizes  $|I_{\alpha}|$
- 3:  $\mathcal{C}_{\alpha} \leftarrow \text{Build-Collection}(\alpha, I_{\alpha})$
- 4: ACTIVATE-COLLECTION( $\mathcal{C}_{\alpha}$ )

These steps are laid out in detail in the next sections. We shall prove the following theorem.

**Theorem 13** (25). Color-Many colors  $\Omega(\ell/d)$  edges in O(m) time.

**Step 1: Choose**  $\alpha$  This step is straightforward.

**Lemma 14.** There is some color  $\alpha$  such that  $|I_{\alpha}| \geq 2\ell/(d+1)$ , and  $\alpha$  can be found in O(m) time.

Proof. Observe that

$$\sum_{\beta \in [d+1]} |I_{\beta}| = \sum_{v \in I} |M(v)| \ge 2\ell$$

Choose  $\alpha$  to be the color missing at the largest number of incomplete vertices. Then  $|I_{\alpha}| \geq 2\ell/(d+1)$ .

We can find  $\alpha$  by counting the number of edges of each color incident to an incomplete vertex (an edge is counted twice if both endpoints are incomplete) and choosing  $\alpha$  to be the color with the lowest count. Finding  $\alpha$  takes O(m) time. Then  $I_{\alpha}$  can be constructed in O(m) time.

Step 2: Build an  $\alpha$ -Collection Color-Many calls a procedure Build-Collection to build an  $\alpha$ -collection  $\mathcal{C}_{\alpha}$  of disjoint fans containing vertices of  $I_{\alpha}$ . This  $\alpha$ -collection shall have the property that each center of a c-fan and each leaf of a u-fan is a vertex of  $I_{\alpha}$  (and the other vertices of  $V(\mathcal{C}_{\alpha})$  are not in  $I_{\alpha}$ ).

BUILD-COLLECTION creates a set S initially containing the vertices of  $I_{\alpha}$ . While S is nonempty, BUILD-COLLECTION removes a vertex v from S and calls a procedure MAKE-COLLECTION-FAN, which attempts to build a primed c-fan with v as the center.

- 1: **procedure** Build-Collection( $\alpha, I_{\alpha}$ )
- 2:  $\mathcal{C}_{\alpha} \leftarrow \emptyset$
- 3:  $S \leftarrow I_{\alpha}$
- 4: for  $v \in S$  do
- 5: Remove v from S
- 6: Choose uncolored incident edge  $vx_0$
- 7:  $F \leftarrow \text{Make-Collection-Fan}(v, x_0, \alpha, \mathcal{C}_{\alpha})$
- 8: If  $F \neq \text{null}$ , add F to  $C_{\alpha}$

Make-Collection-Fan attempts to build a c-fan centered at v in the same way as Make-Primed-Fan, with two important differences: First, if a leaf added to the fan already appears in some other fan of the collection, then the two intersecting fans need to be altered because fans must be disjoint in an  $\alpha$ -collection. This is done by a procedure called Merge-Fans. Second, if a leaf x is missing  $\alpha$ , then the fan is shifted from x immediately and vx is colored by  $\alpha$ . This is necessary to ensure that a vertex of  $I_{\alpha}$  does not end up as a leaf of a c-fan.

Formally, MAKE-COLLECTION-FAN works as follows given an uncolored edge  $vx_0$  with  $v \in I_\alpha$ . Initialize a c-fan  $F = (\alpha, v, x_0)$ . Repeatedly, letting  $x_k$  be the last leaf of F, do the following:

- 1. Check whether  $x_k$  already belongs to a different fan F' in  $\mathcal{C}_{\alpha}$ . If it does, then F and F' overlap at  $x_k$ . Call Merge-Fans(F, F') and return the result.
- 2. If  $\alpha \in M(x_k)$  then shift F from  $x_k$ . Now  $vx_k$  is uncolored and  $\alpha$  is missing at both endpoints. Color  $vx_k$  by  $\alpha$  and discard F. If  $x_k \in S$ , then remove  $x_k$  from S.
- 3. Otherwise, proceed as in MAKE-PRIMED-FAN: Choose a color  $\beta \in M(x_k)$ . If  $\beta \in M(v)$ , then prime F with  $\beta$  and return it. If  $\beta \notin M(v)$ , let  $x_{k+1}$  be the vertex with  $c(vx_{k+1}) = \beta$ . If  $x_{k+1}$  already appears in F, then prime F by  $\beta$  and return F. Otherwise append  $x_{k+1}$  to F as a leaf.

MERGE-FANS(F, F') accepts a c-fan F whose last leaf  $x_k$  also lies in the fan F'. First, it shifts F from  $x_k$  so that  $vx_k$  is uncolored. Then it operates in four cases, depending on whether F' is a c-fan or a u-fan and whether  $x_k$  is the center of F' or a leaf. In all cases, the fan F will be discarded (i.e. not included in the  $\alpha$ -collection). These cases are illustrated in Figure 4.

- Case I: If F' is a c-fan and  $x_k$  is its center, then  $\alpha$  is missing at both v and  $x_k$ . Color  $vx_k$  by  $\alpha$  and discard F'.
- Case II: If F' is a c-fan (with center u) and  $x_k$  is a leaf of F', then create a u-fan as follows: Shift F' from  $x_k$ . Now  $ux_k$  and  $vx_k$  are uncolored and  $\alpha$  is missing at both u and v. By the construction of F',  $\alpha \notin M(x_k)$ . Create a new u-fan having  $x_k$  as its center and u and v as its leaves. Add the u-fan to  $\mathcal{C}_{\alpha}$  and discard F'.
- Case III: If F' is a u-fan and  $x_k$  is its center, then add v as a leaf of F'.
- Case IV: If F' is a u-fan and  $x_k$  is a leaf of F', then  $\alpha$  is missing at both v and  $x_k$ . Color  $vx_k$  by  $\alpha$ . Remove  $x_k$  from F'; if this causes F' to be degenerate (i.e. have less than two leaves) then discard F'.

Analysis of Build-Collection Let us check the correctness and runtime of Build-Collection.

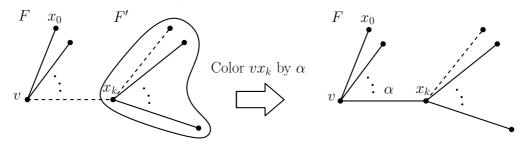
**Lemma 15.** Build-Collection  $(\alpha, I_{\alpha})$  makes an  $\alpha$ -collection in O(m) time.

*Proof.* BUILD-COLLECTION maintains the invariant that every vertex of  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$  is either the center of a c-fan or a leaf of a u-fan. Moreover,  $\alpha$  cannot be missing at any leaf of a c-fan in  $\mathcal{C}_{\alpha}$ . These are ensured by steps 1 and 2 of MAKE-COLLECTION-FAN.

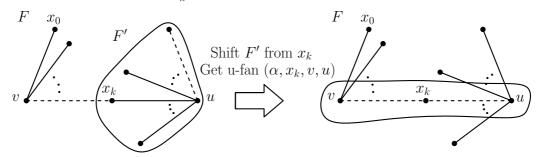
Let us check that  $C_{\alpha}$  remains an  $\alpha$ -collection after a call to MAKE-COLLECTION-FAN. It can terminate in steps 1, 2, or 3.

If it terminates during step 3, then F is a primed c-fan created just as in Make-Primed-Fan and it is disjoint from every other fan in the collection. If it terminates because of step 2, then a

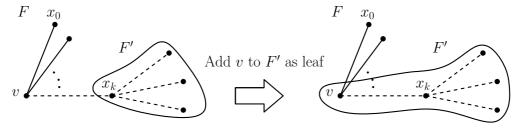
Case I: F' is a c-fan and  $x_k$  is its center.



Case II: F' is a c-fan and  $x_k$  is a leaf.



Case III: F' is a u-fan and  $x_k$  is its center.



Case IV: F' is a u-fan and  $x_k$  is a leaf.

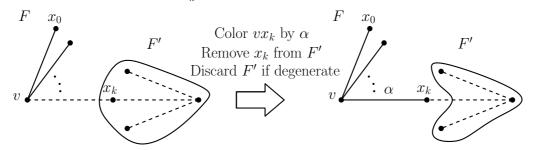


Figure 4: A visual summary of MERGE-FANS. In each of the four cases, a c-fan F intersects at its last leaf  $x_k$  with some fan F' of  $\mathcal{C}_{\alpha}$ . The first step is always to shift F from  $x_k$ . The left side shows the configuration of the intersection (after the shift) and the right side shows the result of MERGE-FANS. The circled vertices in each figure indicate the fans that belongs to  $\mathcal{C}_{\alpha}$ .

new edge is colored and F is discarded. No other fans are affected in these cases, and hence  $\mathcal{C}_{\alpha}$  is an  $\alpha$ -collection.

If it terminates because of step 1, then F is merged with a fan F' in the collection because they

overlap at  $x_k$ . MERGE-FANS only touches the vertices of F and F', so no fans are affected except for these two. It either returns a new u-fan to be added to the collection (Case II), adds a new leaf to an existing u-fan (Case III), or colors a new edge (Cases I and IV), leaving F' in the collection only if it is still a well-formed fan. By inspection of the cases, every fan added to  $\mathcal{C}_{\alpha}$  is well-formed and disjoint from all others in  $\mathcal{C}_{\alpha}$ . Thus, BUILD-COLLECTION outputs an  $\alpha$ -collection.

For the runtime, observe that only the vertices of  $I_{\alpha}$  can ever be the centers of c-fans and at most one c-fan is built around each  $v \in I_{\alpha}$  during Build-Collection. That is, once a vertex of  $I_{\alpha}$  stops being the center of a c-fan, it is never again the center of a c-fan. It takes  $O(\deg(v))$  time to build a c-fan around a center  $v \in I_{\alpha}$ . Hence the total time spent building c-fans is  $O\left(\sum_{v \in I_{\alpha}} \deg(v)\right) = O(m)$ . This accounts for all time spent in steps 2 and 3 of Make-Collection-Fan.

The remaining time is spent on Merge-Fans. Every operation in Merge-Fans takes constant time except for shifting F and, in Case II, shifting F'. In all cases, any shifted fan is always discarded immediately afterwards, and so each c-fan that is built can be shifted only once. The total time for all shifts is therefore bounded by the sum of sizes of all c-fans:  $O\left(\sum_{v\in I_{\alpha}} \deg(v)\right) = O(m)$ . Excluding the time spent on shifts, Merge-Fans takes constant time each time it is called (and it is called at most  $I_{\alpha}$  | times). Hence Build-Collection takes O(m) time.

Now let us estimate the number of edges colored during BUILD-COLLECTION and the number of fans in  $\mathcal{C}_{\alpha}$ . There is be a balance between these two quantities because some vertices of  $I_{\alpha}$  do not end up in  $V(\mathcal{C}_{\alpha})$ , but that only happens in cases where some edge is colored.

**Lemma 16.** Let  $\ell_b$  and  $\ell_m$ , respectively, be the number of uncolored edges before and after Build-Collection. Let  $\mathcal{C}_{\alpha}$  be the collection output by Build-Collection $(I_{\alpha})$ . Then

$$|V(\mathcal{C}_{\alpha}) \cap I_{\alpha}| + 3(\ell_b - \ell_m) \ge |I_{\alpha}|$$

*Proof.* Let  $v \in I_{\alpha}$ . There are only a few ways that BUILD-COLLECTION can affect v.

Possibly, v could end up in some fan of  $\mathcal{C}_{\alpha}$ , either as the center of a c-fan or a leaf of a u-fan. That is, v could be in  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$ .

If this does not happen, it is because v is part of a discarded fan or removed from its fan. These events can occur during the construction of a c-fan (in step 2 of MAKE-COLLECTION-FAN) or in Cases I and IV of MERGE-FANS. However, all three of these scenarios coincide with an uncolored edge being colored. Observe that every time an edge is colored, at most three<sup>9</sup> vertices are removed from  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$ .

Therefore, at most  $3(\ell_b - \ell_m)$  vertices of  $I_\alpha$  do not end up in a fan of  $\mathcal{C}_\alpha$ . It follows that

$$|V(\mathcal{C}_{\alpha}) \cap I_{\alpha}| + 3(\ell_b - \ell_m) \ge |I_{\alpha}|$$

Step 3: Activate the  $\alpha$ -Collection Once  $\mathcal{C}_{\alpha}$  is constructed, Color-Many calls Activate-Collection to color a large fraction of the uncolored edges in the fans of  $\mathcal{C}_{\alpha}$ . Together with the edges colored in step 2,  $\Omega(\ell/d)$  edges will be colored in total.

ACTIVATE-COLLECTION works in d stages, one for each color other than  $\alpha$ . Let  $\beta_1, \ldots, \beta_d$  be the colors of  $[d+1] \setminus {\alpha}$  in some fixed order. For  $i=1,\ldots,d$ , let

$$Q_i = \{ F \in \mathcal{C}_{\alpha} \mid F \text{ is a c-fan primed by } \beta_i \text{ or } F \text{ is a u-fan and } \beta_i \text{ is missing at its center} \}$$

<sup>&</sup>lt;sup>9</sup>The worst case happens in Case IV when F' is a u-fan with two leaves, one being  $x_k$ . The edge  $vx_k$  is colored and v and the two leaves of F' are discarded.

We assume that the algorithm correctly maintains all the sets  $Q_i$  at all times. This can be handled automatically with constant overhead when coloring and uncoloring edges.

ACTIVATE-COLLECTION runs in d stages, where the ith stage deals only with the fans of  $Q_i$ . Stage i works as follows. While  $Q_i$  is nonempty, pick a fan  $F \in Q_i$  and let v be the center of F. If F is a c-fan (primed by  $\beta_i$ ) then call ACTIVATE-C-FAN(F). If F is a u-fan (with  $\beta_i \in M(v)$ ) then call ACTIVATE-U-FAN $(F, \beta_i)$ . After the activation, discard F if it is a c-fan or a degenerate u-fan. Then call DISCONNECT-VERTEX(v), a procedure that cleans up the damage that may have been caused by the activation.

ACTIVATE-COLLECTION is given in pseudocode below.

```
1: procedure ACTIVATE-COLLECTION(Q_1, \ldots, Q_d)
        for i from 1 to d do
2:
               \{Stage\ i\}
3:
             while Q_i \neq \emptyset do
4:
                 Pick F \in Q_i
5:
                 if F = (\alpha, v, x_0, \dots, x_k) is a c-fan primed by \beta_i then
6:
                     Activate-c-Fan(F)
7:
                     Remove F from \mathcal{C}_{\alpha}
8:
                 else \{F = (\alpha, v, x_1, \dots, x_k) \text{ is a u-fan and } \beta_i \in M(v)\}
9:
                     ACTIVATE-U-FAN(F, \beta_i)
10:
                     if F has less than two leaves then
11:
                         Remove F from \mathcal{C}_{\alpha}
12:
                 DISCONNECT-VERTEX(v)
13:
```

Except for the call to DISCONNECT-VERTEX, this procedure doesn't do anything surprising. It activates each fan using the appropriate function, and then discards it unless it is a u-fan that could be activated again later. The purpose of DISCONNECT-VERTEX is to address two issues.

First, if the alternating path that was flipped during the activation ends at a vertex w of some fan F', then F' may no longer be well-formed (i.e. may not adhere to the definition of a u-fan or a primed c-fan) because M(w) has changed — specifically,  $\alpha$  and  $\beta_i$  have been exchanged in M(w). Whenever a fan F' is malformed for this reason, we say F' is damaged and that w is the damaged vertex of F'.

A damaged fan must be repaired or discarded before the next activation, for otherwise  $C_{\alpha}$  contains some "fan-like object" that does not satisfy the definition of a fan. Let us formalize this requirement.

**Precondition 1:**  $C_{\alpha}$  is always an  $\alpha$ -collection before a fan is activated during ACTIVATE-COLLECTION. In particular, no fan in  $C_{\alpha}$  is damaged.

Second, the edges of the flipped path may now belong to some larger  $\alpha\beta_i$ -path in the graph which could be flipped later during a different activation. If the same edges are in many different flipped paths during stage i, the runtime could exceed O(m). We avoid this by isolating every vertex in the flipped path so that it cannot be accessed again during stage i.

During stage i, say a vertex x is disconnected if it lies in an  $\alpha\beta_i$ -path whose endpoints are not in  $V(\mathcal{C}_{\alpha})$ .

**Precondition 2:** Before a fan is activated during stage i, any vertex that had previously been in a flipped path during stage i is disconnected.

Note that DISCONNECT-VERTEX may also flip  $\alpha\beta_i$ -paths, and the vertices in those paths must satisfy Precondition 2 as well.

**Disconnect-Vertex** The purpose of DISCONNECT-VERTEX is to ensure that these preconditions are satisfied at the top of each iteration. DISCONNECT-VERTEX is given a vertex v and it explores the  $\alpha\beta_i$ -path P containing it. It manipulates the fans containing the endpoints of the path with the twofold goal of repairing those fans if they are damaged (with the endpoint being the damaged vertex) and ensuring that the endpoints of P do not lie in fans of  $C_{\alpha}$ .

The procedure works as follows. Let P be the  $\alpha\beta_i$ -path containing v. Until neither endpoint of P lies in a fan of  $\mathcal{C}_{\alpha}$ , do the following: Choose w to be an endpoint of P that lies in a fan  $F' \in \mathcal{C}_{\alpha}$ , and if possible choose w so that F' is damaged (with w being the damaged vertex). Then act in three cases.

- Case 1: If F' is a c-fan or a u-fan with only two leaves, then discard F' (i.e. delete it from  $\mathcal{C}_{\alpha}$ ).
- Case 2: If F' is a u-fan with at least three leaves and w is a leaf of F', then remove w from F'.
- Case 3: If F' is a u-fan with at least three leaves and w is the center of F', then choose a leaf x of F'. If  $x \in P$ , replace x with a different leaf of F'.

Now we have two subcases depending on whether  $\alpha \in M(w)$ .

- Case 3a: If  $\alpha \in M(w)$ , then w must be the damaged vertex of F'. We have  $\alpha \in M(w) \cap M(x)$ . Color wx by  $\alpha$  and remove x from F'. Now F' is undamaged.
- Case 3b: If  $\alpha \notin M(w)$ , then F' is a valid u-fan. We must have  $\beta_i \in M(w)$  for otherwise w would not be an endpoint of P. Flip the  $\alpha\beta_i$ -path beginning at x. Now  $\beta_i \in M(w) \cap M(x)$ . Color wx by  $\beta_i$  and remove x from F'.

Now wx is colored by  $\alpha$  or  $\beta_i$ , and so w is no longer an endpoint of P. Extend P by continuing the  $\alpha\beta_i$ -path through wx until it reaches an endpoint.

DISCONNECT-VERTEX is given below in pseudocode.

```
1: procedure DISCONNECT-VERTEX(v)
2:
        Let P be the \alpha\beta_i-path containing v
        while an endpoint of P lies in a fan of \mathcal{C}_{\alpha} do
3:
4:
           Let e_1 and e_2 be the endpoints of P
           if e_1 is the damaged vertex of a fan or e_2 is not in a fan then
5:
6:
               w \leftarrow e_1
           else
7:
8:
               w \leftarrow e_2
           Let F' be the fan containing w
9:
   Case 1
           if F' is a c-fan or a u-fan with two leaves then
10:
               Remove F' from \mathcal{C}_{\alpha}
11:
           else
12:
   Case 2
               if F' is a u-fan and w is a leaf of F' then
13:
                   Remove w from F'
14:
               else
15:
   Case 3
                   if F' is a u-fan and w is the center of F' then
16:
                       Choose a leaf x of F'
17:
                       if x \in P then
18:
```

```
Replace x with a different leaf of F'
19:
                          Remove x from F'
20:
                          if \alpha \in M(w) then
21:
                   Case 3a
                                 \{F' \text{ is damaged as } \alpha \in M(w)\}
22:
                                 \{\alpha \in M(w) \cap M(x)\}
23:
                               Color wx by \alpha
24:
                          else
25:
                   Case 3b
                                 \{F' \text{ is not damaged}\}\
26:
                              Flip the \alpha\beta_i-path containing x
27:
                                \{\beta_i \in M(w) \cap M(x)\}
28:
                              Color wx by \beta_i
29:
                            \{w \text{ is no longer an endpoint of } P\}
30:
                          Extend P by following the \alpha\beta_i-path through wx
31:
```

Analysis of Disconnect-Vertex Let us first establish a few facts about Disconnect-Vertex.

**Proposition 17.** Consider a fan F with center v that is activated during stage i of ACTIVATE-COLLECTION. Let  $P_F$  be the  $\alpha\beta_i$ -path flipped during the activation and let v and w be its endpoints, and assume Preconditions 1 and 2 held before the activation of F. At all times during DISCONNECT-VERTEX(v), the following hold.

- 1. P contains  $P_F$ .
- 2. P is a maximal  $\alpha\beta_i$ -path. In particular, P never becomes a cycle.
- 3. At most one endpoint of P is the damaged vertex of a fan in  $C_{\alpha}$  and no other fan in  $C_{\alpha}$  is damaged.

*Proof.* We check these in order.

- 1. At the start of DISCONNECT-VERTEX(v), P contains  $P_F$ . This can be checked by case analysis of ACTIVATE-C-FAN and ACTIVATE-U-FAN. Since vertices are never removed from P, it remains true.
- 2. Initially, P is chosen to be the (maximal)  $\alpha\beta_i$ -path containing v. Suppose P is a maximal  $\alpha\beta_i$ -path at the start of the loop (at line 4 in DISCONNECT-VERTEX). At the end of the loop, either P will be unchanged and will still be maximal (as in Cases 1 and 2), or P will have been extended at one endpoint by Case 3. In all cases, P will be maximal.
  - We must check that P cannot become a cycle due to Case 3. Case 3 extends P through an edge wx of a u-fan F', where w is the center of F' (and an endpoint of P) and x is a leaf of F'. Observe that any leaf of F' that lies in P must be the other endpoint of P, since  $\alpha$  is missing at each leaf. If necessary, the procedure replaces x by a different leaf so that x is not in P. Thus, when wx is colored to join P with the  $\alpha\beta_i$ -path containing x, it cannot create a cycle.
- 3. Before DISCONNECT-VERTEX is called, only the fan containing w (if any) can be damaged and only with w being the damaged vertex. Indeed, M(w) has been altered because  $P_F$  was flipped, and so that fan may not satisfy the definition of a u-fan or primed c-fan. We

check that all other fans in  $\mathcal{C}_{\alpha}$  are still well-formed. The activation of F only affected fans containing vertices of  $P_F$  (including F itself). The fans containing only *interior* vertices of  $P_F$  cannot be made invalid due to the activation of F because flipping  $P_F$  does not change the set of missing colors at any interior vertex. The fan F is discarded immediately unless it is a non-degenerate u-fan after its activation, in which case it is still well-formed. Thus, only the fan containing w can be malformed, and only in the sense that w is the damaged vertex of that fan.

Now assume inductively that at most one endpoint of P is the damaged vertex of a fan in  $\mathcal{C}_{\alpha}$  and all other fans are well-formed. If an endpoint of P is the damaged vertex of a fan F', then algorithm chooses w to be that vertex. In Case 1, F' is discarded and so all fans left in  $\mathcal{C}_{\alpha}$  are well-formed. In Case 2, w is removed from F' which repairs F', and w is no longer in any fan of  $\mathcal{C}_{\alpha}$ . In Case 3a, F' is damaged initially and repaired by coloring an incident edge by  $\alpha$ ; at this point, no fan can be damaged. After these cases, no fan in  $\mathcal{C}_{\alpha}$  can be damaged.

In Case 3b, F' is not damaged and so no fan in  $\mathcal{C}_{\alpha}$  is damaged, for otherwise w would have been chosen to be the damaged vertex. However, when the  $\alpha\beta_i$ -path containing x is flipped, the fan on the other end could be damaged. The damaged vertex of this fan becomes the new endpoint of P when P is extended through the edge wx. Thus, there is at most one damaged fan at the end of this case and its damaged vertex must be an endpoint of P.

Now we can check that DISCONNECT-VERTEX ensures Preconditions 1 and 2. Establishing Precondition 1 will imply the correctness of Activate-Collection, and Precondition 2 will be useful in measuring its runtime.

**Lemma 18.** Preconditions 1 and 2 hold before each fan is activated in Activate-Collection.

*Proof.* Both preconditions hold before any fan is activated. Assume inductively that the preconditions hold before the activation of a fan F.

By Proposition 17, no fan can be damaged or otherwise malformed except for those containing the endpoints of P, and neither endpoint of P lies in a fan of  $\mathcal{C}_{\alpha}$  when DISCONNECT-VERTEX(v) terminates. Clearly DISCONNECT-VERTEX must terminate as each iteration decreases  $|V(\mathcal{C}_{\alpha})|$ . Therefore Precondition 1 is restored.

Now we check Precondition 2. By Proposition 17, P is always maximal  $\alpha\beta_i$ -path and it contains all the vertices of the path flipped during the activation of F. When the procedure is over, every vertex in P is disconnected because the endpoints of P are not in fans of  $\mathcal{C}_{\alpha}$ .

All other vertices that were disconnected before F was activated must still be disconnected. This holds because every edge colored by  $\alpha$  or  $\beta_i$  that is affected by the activation of F and the subsequent call to Disconnecte-Vertex becomes an edge in P. Hence Precondition 2 is restored.

**Lemma 19.** The paths  $P_1, \ldots, P_k$  traversed by the applications of DISCONNECT-VERTEX during stage i are vertex-disjoint.

Proof. Proceed by induction on k. Let  $P_k$  be the  $\alpha\beta_i$ -path constructed by DISCONNECT-VERTEX after the activation of a fan F with center v. By Precondition 2, all the vertices in  $P_1, \ldots, P_{k-1}$  are disconnected before the activation. Since v was the center of F, v was not disconnected before the activation, and so it cannot lie in  $P_1, \ldots, P_{k-1}$ . At the start of DISCONNECT-VERTEX(v),  $P_k$  is the  $\alpha\beta_i$ -path containing v. Whenever DISCONNECT-VERTEX extends  $P_k$  by coloring an edge (as in Case 3), the  $\alpha\beta_i$ -path that forms the extension has an endpoint in a fan of  $\mathcal{C}_{\alpha}$ . Hence a path  $P_j$  (j < k) cannot be traversed during the construction of  $P_k$ , and so  $P_k$  is disjoint from all previous paths. It follows that  $P_1, \ldots, P_k$  are disjoint.

Finally, we measure the runtime of DISCONNECT-VERTEX.

**Lemma 20.** DISCONNECT-VERTEX takes O(length(P)) time, where P is the constructed path.

*Proof.* The runtime is clear as constant time is spent on each vertex of P and vertices are never removed from P.

Analysis of Activate-Collection We aim to prove that ACTIVATE-COLLECTION runs in O(m) time and colors a number of edges that is at least a constant fraction of  $|V(\mathcal{C}_{\alpha}) \cap I_{\alpha}|$ .

**Lemma 21.** ACTIVATE-COLLECTION runs in O(m) time.

Proof. Consider a fan F in  $C_{\alpha}$ . If F is a c-fan, then it takes O(|F| + length(P)) time to activate F and call DISCONNECT-VERTEX, where P is the entire  $\alpha\beta_i$ -path formed by DISCONNECT-VERTEX. This follows from Lemma 20 and the fact that the path flipped during the activation of F is contained in P. If F is a u-fan, then the time is just O(length(P)) because the activation itself (not including the flipping operation) takes constant time.

Let  $P_1, \ldots, P_T$  be the paths explored by DISCONNECT-VERTEX during ACTIVATE-COLLECTION. Choose indices  $1 = T_0 \le T_1 \le \cdots \le T_d = T+1$  such that  $P_{T_{i-1}}, \ldots, P_{T_i-1}$  are the paths constructed during stage i. The total time is then

$$O\left(\sum_{\text{c-fan }F\in\mathcal{C}_{\alpha}}|F|+\sum_{i=1}^{d}\sum_{t=T_{i-1}}^{T_{i}-1}\operatorname{length}(P_{t})\right)$$

We have  $\sum_{c-\text{fan }F\in\mathcal{C}_{\alpha}}|F|\leq m$  as the c-fans are disjoint and a c-fan can only be activated once. We now show that the sum of the lengths of all paths  $P_t$  is O(m).

For  $i=1,\ldots,d$ , let  $m_i$  be the number of edges colored by  $\beta_i$  at the end of stage i. Observe that the ith stage only modifies the number of edges colored by  $\alpha$  and  $\beta_i$ —all others are fixed. Indeed, the only operation in stage i that involves an edge not colored by  $\alpha$  or  $\beta$  is the shift operation, and shifting a c-fan does not change the number of edges of each color. That means, for each i, the number of edges colored by  $\beta_i$  after stage d is also  $m_i$ . Therefore  $\sum_{i=1}^d m_i \leq m$ .

Now consider stage i. By Lemma 19,  $P_{T_{i-1}}, \ldots, P_{T_i-1}$  are disjoint. At the end of stage i, each path  $P_t$  that was built during stage i is an  $\alpha\beta_i$ -path, and so has at most one more edge colored by  $\alpha$  than by  $\beta_i$ . That is, if  $P_t$  has  $k_t$  edges colored by  $\beta_i$  then length( $P_t$ )  $\leq 2k_t + 1$ . Thus,

$$\sum_{t=T_{i-1}}^{T_i-1} \operatorname{length}(P_t) \le \sum_{t=T_{i-1}}^{T_i-1} (2k_t+1) \le 2m_i + (T_i - T_{i-1})$$

Summing this estimate over all rounds gives us

$$\sum_{i=1}^{d} \sum_{t=T_{i-1}}^{T_i-1} \operatorname{length}(P_t) \le \sum_{i=1}^{d} 2m_i + (T_i - T_{i-1}) = 2\left(\sum_{i=1}^{d} m_i\right) + (T_d - T_0) \le 2m + T \le 3m$$

The last inequality holds because every activation colors a new edge, and so  $T \leq m$ . Therefore, ACTIVATE-COLLECTION runs in O(m) time.

Now we establish that ACTIVATE-COLLECTION does not ignore any fans in  $\mathcal{C}_{\alpha}$ .

**Lemma 22.**  $\mathcal{C}_{\alpha}$  is empty after ACTIVATE-COLLECTION( $\mathcal{C}_{\alpha}$ ).

Proof. ACTIVATE-COLLECTION runs in d stages, where the ith stage finishes once  $Q_i$  is empty. We show that no fan can be added to  $Q_i$  in later stages. A c-fan cannot be added to  $Q_i$  in later stages because the color that primes a c-fan never changes. A u-fan  $F \notin Q_i$  can only be added to  $Q_i$  when its center is the endpoint of a flipped path that interchanges  $\alpha$  and  $\beta_i$ . However,  $\alpha\beta_i$ -paths are only flipped during stage i, and so u-fans can only be added to  $Q_i$  during stage i. Thus,  $Q_i$  remains empty after stage i. It follows that  $Q_1, \ldots, Q_d$  are all empty at the end of ACTIVATE-COLLECTION.

Next, we compare the number of edges colored by ACTIVATE-COLLECTION against the number of vertices removed from  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$ .

**Lemma 23.** In every iteration of ACTIVATE-COLLECTION, the number of uncolored edges decreases by  $r \geq 1$  and at most r + 6 vertices of  $I_{\alpha}$  are removed from  $V(C_{\alpha})$ .

*Proof.* Recall that the center of a c-fan is in  $I_{\alpha}$  (but none of the leaves are) and each leaf of a u-fan is in  $I_{\alpha}$  (but the center is not). Activating a fan always causes a new edge to be colored and it removes a vertex of  $I_{\alpha}$  from that fan: either the center of a c-fan or a leaf of a u-fan. If it is a u-fan then another vertex of  $I_{\alpha}$  may be removed if the flipped path ends at a leaf of that fan, and a third vertex of  $I_{\alpha}$  may be removed if the u-fan becomes degenerate and is discarded. Thus, due to the activation, one new edge is colored and  $|V(\mathcal{C}_{\alpha}) \cap I_{\alpha}|$  decreases by at most 3.

ACTIVATE-COLLECTION then calls DISCONNECT-VERTEX, which may color more edges and remove more vertices of  $I_{\alpha}$ . In the first two cases of DISCONNECT-VERTEX, no edges are colored and at most two vertices of  $I_{\alpha}$  are removed from  $V(\mathcal{C}_{\alpha})$ . These cases can happen only twice during DISCONNECT-VERTEX because they cause an endpoint of P to not lie in any fan of  $\mathcal{C}_{\alpha}$ . Hence, these cases cause  $|V(\mathcal{C}_{\alpha}) \cap I_{\alpha}|$  to decrease by at most 4. Case 3 can occur many times, but it always colors one edge (either by  $\alpha$  or by  $\beta_i$ ) and removes one vertex of  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$ .

Thus, if r new edges are colored during an iteration of ACTIVATE-COLLECTION, at most r+6 vertices are removed from  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$ : Up 3 vertices are lost due the activation of the fan (while 1 new edge is colored) and up to (r-1)+4 vertices are lost during DISCONNECT-VERTEX (while r-1 new edges are colored).

The next corollary counts the number of edges newly colored by ACTIVATE-COLLECTION( $\mathcal{C}_{\alpha}$ ).

Corollary 24. Let  $\ell_m$  and  $\ell_f$ , respectively, be the number of uncolored edges before and after ACTIVATE-COLLECTION. Let  $\mathcal{C}_{\alpha}$  be the collection output by BUILD-COLLECTION. Then

$$\ell_m - \ell_f \ge \frac{|V(\mathcal{C}_\alpha) \cap I_\alpha|}{7}$$

*Proof.* By Lemma 22,  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$  is empty at the end of ACTIVATE-COLLECTION. By Lemma 23, at least one edge is colored for every 7 vertices removed from  $V(\mathcal{C}_{\alpha}) \cap I_{\alpha}$ .

We are now able to wrap up the proof of Theorem 25.

**Theorem 25.** Color-Many colors  $\Omega(\ell/d)$  edges in O(m) time.

*Proof.* Color-Many runs in O(m) time by Lemmas 14, 15, and 21.

Let  $\ell_b$ ,  $\ell_m$ , and  $\ell_f$ , respectively, be the number of uncolored edges at the start of Color-Many, after Build-Collection, and after Activate-Collection. Recall that  $|I_{\alpha}| \geq 2\ell_b/(d+1)$  by Lemma 14. By Lemma 16,

$$|V(\mathcal{C}_{\alpha}) \cap I_{\alpha}| + 3(\ell_b - \ell_m) \ge |I_{\alpha}|$$

By Corollary 24,

$$\ell_m - \ell_f \ge \frac{|V(\mathcal{C}_\alpha) \cap I_\alpha|}{7}$$

Hence,

$$\ell_b - \ell_f = (\ell_b - \ell_m) + (\ell_m - \ell_f)$$

$$\geq \frac{|I_\alpha| - |V(\mathcal{C}_\alpha) \cap I_\alpha|}{3} + \frac{|V(\mathcal{C}_\alpha) \cap I_\alpha|}{7}$$

$$= \frac{7|I_\alpha| - 4|V(\mathcal{C}_\alpha) \cap I_\alpha|}{21}$$

$$\geq \frac{7|I_\alpha| - 4|I_\alpha|}{21}$$

$$= \frac{|I_\alpha|}{7}$$

$$\geq \frac{2\ell_b}{7(d+1)}$$

Therefore,  $\Omega(\ell/d)$  edges are colored by Color-Many.

## References

[1] Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. Switch scheduling via randomized edge coloring. In 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings., pages 502–512. IEEE, 2003.

- [2] Noga Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Information Processing Letters*, 85(6):301–302, 2003.
- [3] Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with  $\delta + 1$  colours. INFOR: Information Systems and Operational Research, 20(2):82–101, 1982.
- [4] Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. Synthesis Lectures on Distributed Computing Theory, 4(1):1–171, 2013.
- [5] George Bennett. Probability inequalities for the sum of independent random variables. *Journal* of the American Statistical Association, 57(297):33–45, 1962.
- [6] S Bernstein. Sur une modification de l'inéqualité de tchebichef. Annals Science Institue Sav. Ukraine Sect. Math. I, pages 38–49, 1924.
- [7] 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.
- [8] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2830–2842. SIAM, 2021.
- [9] Maggie Cheng and Li Yin. Transmission scheduling in sensor networks via directed edge coloring. In 2007 IEEE International Conference on Communications, pages 3710–3715. IEEE, 2007.

- [10] Marek Chrobak and Takao Nishizeki. Improved edge-coloring algorithms for planar graphs. Journal of Algorithms, 11(1):102–116, 1990.
- [11] Richard Cole and Łukasz Kowalik. New linear-time algorithms for edge-coloring planar graphs. *Algorithmica*, 50(3):351–368, 2008.
- [12] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in  $O(E \log D)$  time. Combinatorica, 21(1):5–12, 2001.
- [13] Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1937–1945. SIAM, 2019.
- [14] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, 2005.
- [15] Harold N Gabow. Using euler partitions to edge color bipartite multigraphs. *International Journal of Computer & Information Sciences*, 5(4):345–355, 1976.
- [16] Harold N Gabow and Oded Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. SIAM journal on Computing, 11(1):117–129, 1982.
- [17] Harold N Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. Algorithms for edge-coloring. Technical report, Technical report 41/85, Tel Aviv University, 1985.
- [18] Shashidhar Gandham, Milind Dawande, and Ravi Prakash. Link scheduling in wireless sensor networks: Distributed edge-coloring revisited. *Journal of Parallel and Distributed Computing*, 68(8):1122–1134, 2008.
- [19] Dorit S Hochbaum, Takao Nishizeki, and David B Shmoys. A better than "best possible" algorithm to edge color multigraphs. *Journal of Algorithms*, 7(1):79–104, 1986.
- [20] Ian Holyer. The NP-completeness of edge-coloring. SIAM Journal on computing, 10(4):718–720, 1981.
- [21] Murali Kodialam and Thyaga Nandagopal. Characterizing achievable rates in multi-hop wireless networks: the joint routing and scheduling problem. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 42–54. ACM, 2003.
- [22] Peter Sanders and David Steurer. An asymptotic approximation scheme for multigraph edge coloring. ACM Transactions on Algorithms (TALG), 4(2):21, 2008.
- [23] Claude E Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949.
- [24] Vadim G Vizing. On an estimate of the chromatic class of a p-graph. Discret Analiz, 3:25–30, 1964.
- [25] David P Williamson, Leslie A Hall, Jan A Hoogeveen, Cor AJ Hurkens, Jan Karel Lenstra, Sergey Vasil'evich Sevast'janov, and David B Shmoys. Short shop schedules. *Operations Research*, 45(2):288–294, 1997.

# A Dictionary Data Structure

Here we describe the implementation of  $\mathcal{D}$ , the dictionary data structure introduced in Section 6.4. Recall that  $\mathcal{D}$  is meant to map a vertex-color pair  $(v, \gamma)$  to the edge incident to v colored by  $\gamma$ , if such an edge exists, and we wish to support SEARCH, INSERT, and DELETE in constant time.

For this description, treat the vertices of G as integers from 0 to n-1. Intuitively, we would like to simply store an array of length n(d+1) where the entry at index  $(v(d+1)+\gamma)$  contains the edge incident to v colored by  $\gamma$ . This may be too costly in preprocessing as n(d+1) could exceed our desired time complexity of  $O(m\sqrt{n})$ . Instead we use a two-level decomposition of this method which takes  $O(\sqrt{mnd})$  time to initialize.

 $\mathcal{D}$  must store key-value pairs, where the keys are drawn from a universe  $V \times [d+1]$  of size U = n(d+1) and we know that the dictionary contains at most M = 2m entries at any time (at most one for each endpoint).

Let  $b = \lceil \sqrt{U/M} \rceil$ . Divide the universe into  $\lceil U/b \rceil$  ranges  $R_0, \ldots, R_{\lceil U/b \rceil - 1}$  of size at most b (specifically, the ranges are [0, b - 1], [b, 2b - 1], etc). Initialize an array A of length  $\lceil U/b \rceil$ , where the kth entry of A corresponds to the range  $R_k$ . For each index k of A, store a counter C[k] that tracks the number of entries in range  $R_k$ .

Then initialize M arrays  $B_1, B_2, \ldots, B_M$  of length b. Call these block arrays. An entry of a block array is either empty (**null**) or it has a pointer to an edge.

If C[k] = 0, then A[k] is empty (it contains **null**). Otherwise, A[k] contains a pointer to a block array B, and for as long as C[k] > 0, B will act as an array storing the entries in the range  $R_k$ . At any time, B can only be linked from one location in A (but it can change locations over time).

Create a linked list E which shall contain the empty (unused) block arrays. Initially all block arrays are in E.

The operations are implemented as follows:

- SEARCH $(v, \gamma)$ : Write the index  $v(d+1) + \gamma$  as a sum kb + j with k and j integers.<sup>10</sup> If A[k] = null then return null. If A[k] = B, then return B[j].
- INSERT $(v, \gamma, e)$ : Compute k and j as above. If A[k] = B then set  $B[j] = \gamma$ . If A[k] = null, then remove an empty block array B from E, set A[k] = B, and set  $B[j] = \gamma$ . Increment C[k].

Observe that E cannot be empty when a block array is removed from E, for there are M block arrays and at most M entries in the dictionary.

• DELETE $(v, \gamma)$ : Compute k and j as above. If  $A[k] = \mathbf{null}$ , then do nothing. If A[k] = B and  $B[j] \neq \mathbf{null}$ , then set  $B[j] = \mathbf{null}$  and decrement C[k]. If C[k] = 0 now, then set  $A[k] = \mathbf{null}$  and append B to E because B is empty.

Each operation takes constant time. Initializing the data structure takes  $O(U/b + Mb) = O(\sqrt{UM}) = O(\sqrt{mnd})$  time.

A Note About Alternative Implementations Our implementation is somewhat unconventional, and it would be nice if an out-of-the-box dictionary could be used instead. Our goal is to have constant time operations, and the barrier in our case is just the preprocessing time. We need  $\mathcal{D}$  to be initialized in  $O(m\sqrt{n})$  time. There are alternative solutions available.

Firstly, if  $m\sqrt{n} = \Omega(nd)$ , as is often the case, we can simply use an array of length n(d+1) as our dictionary.

<sup>&</sup>lt;sup>10</sup>Specifically  $k = \lfloor (v(d+1) + \gamma)/b \rfloor$  and  $j = (v(d+1) + \gamma) \mod b$ .

Secondly, there are *randomized* dictionaries that can be initialized in constant time and perform all operations in constant expected amortized time (e.g. generalized cuckoo hashing [14]). If deterministic computation is not a priority, and certainly in the case of RANDOM-EULER-COLOR, this type of dictionary can be substituted.

In fact,  $\mathcal{D}$  is only truly needed for RANDOM-COLOR-ONE. Versions of COLOR-ONE and COLOR-MANY exist that do not use  $\mathcal{D}$  at all; this was the case for RECOLOR and PARALLEL-COLOR, the analogous subroutines in [17]. Although  $\mathcal{D}$  is a valuable tool that simplifies their implementations, it is possible to achieve the same deterministic  $O(m\sqrt{n})$  bound for edge-coloring without it.

The Recursive Strategy and  $\mathcal{D}$  We must be careful in how we manage our data structures with respect to the recursive strategy. Since  $\mathcal{D}$  takes more than O(m) time to initialize, we cannot afford to reinitialize  $\mathcal{D}$  for each subgraph during EULER-COLOR or RANDOM-EULER-COLOR. If done improperly, different branches of the recursion could run into conflicts by reading and writing to the same locations in  $\mathcal{D}$ , even though they are operation on edge-disjoint subgraphs.

A simple workaround is to populate and depopulate  $\mathcal{D}$  in every Repair step. Before the Repair step begins, insert the color of every edge into  $\mathcal{D}$ ; when it is over, remove every color from  $\mathcal{D}$ . This way, every time a Repair step occurs during the strategy,  $\mathcal{D}$  exactly reflects the current subgraph and its colors. It adds O(m) insertions and deletions to every Repair step which does not change the asymptotic runtime. It is also possible to do without this workaround by careful management of the colors.