Path-Coloring Algorithms for Plane Graphs

Aven Bross
Department of Computer Science
University of Alaska
Fairbanks, AK 99775-6670
dabross@alaska.edu

Glenn G. Chappell
Department of Computer Science
University of Alaska
Fairbanks, AK 99775-6670
chappellg@member.ams.org

Chris Hartman
Department of Computer Science
University of Alaska
Fairbanks, AK 99775-6670
cmhartman@alaska.edu

December 5, 2024

2010 Mathematics Subject Classification. Primary 05C38; Secondary 05C10, 05C15.
Key words and phrases. Path coloring, list coloring, algorithm.

Abstract

A path coloring of a graph G is a vertex coloring of G such that each color class induces a disjoint union of paths. We present two efficient algorithms to construct a path coloring of a plane graph.

The first algorithm, based on a proof of Poh, is given a plane graph; it produces a path coloring of the given graph using three colors.

The second algorithm, based on similar proofs by Hartman and Škrekovski, performs a list-coloring generalization of the above. The algorithm is given a plane graph and an assignment of lists of three colors to each vertex; it produces a path coloring of the given graph in which each vertex receives a color from its list.

Implementations of both algorithms are available.

1 Introduction

All graphs will be finite, simple, and undirected. See West [10] for graph theoretic terms.

A *path coloring* of a graph G is a vertex coloring (not necessarily proper) of G such that each color class induces a disjoint union of paths. A graph G is *path k-colorable* if G admits a path coloring using k colors.

Broere & Mynhardt conjectured [2, Conj. 16] that every planar graph is path 3-colorable. This was proven independently by Poh [8, Thm. 2] and by Goddard [6, Thm. 1].

Theorem 1.1 (Poh 1990, Goddard 1991)**.** *If G is a planar graph, then G is path* 3*-colorable.* □

It is easily shown that the "3" in Theorem 1.1 is best possible. In particular, Chartrand & Kronk [5, Section 3] gave an example of a planar graph whose vertex set cannot be partitioned into two subsets, each inducing a forest.

Hartman [7, Thm. 4.1] proved a list-coloring generalization of Theorem 1.1 (see also Chappell & Hartman [4, Thm. 2.1]). A graph G is *path k-choosable* if, whenever each vertex of G is assigned a list of k colors, there exists a path coloring of G in which each vertex receives a color from its list.

Theorem 1.2 (Hartman 1997)**.** *If G is a planar graph, then G is path* 3*-choosable.* □

Essentially the same technique was used by Škrekovski [9, Thm. 2.2b] to prove a result slightly weaker than Theorem 1.2.

We discuss two efficient path-coloring algorithms based on proofs of the above theorems. We distinguish between a *planar* graph—one that can be drawn in the plane without crossing edges—and a *plane* graph—a graph with a given embedding in the plane.

In Section 2 we outline our graph representations and the basis for our computations of time complexity.

Section 3 covers an algorithm based on Poh's proof of Theorem 1.1. The algorithm is given a plane graph; it produces a path coloring of the given graph using three colors.

Section 4 covers an algorithm based Hartman's proof of Theorem 1.2, along with the proof of Škrekovski mentioned above. The algorithm is given a plane graph and an assignment of a list of three colors to each vertex; it produces a path coloring of the given graph in which each vertex receives a color from its list.

Section 5 provides benchmark results for implementations of both algorithms [3]. The section also discusses how each algorithm may be modified to benefit from parallelism.

2 Graph Representations and Time Complexity

We will represent a graph via *adjacency lists*: a list, for each vertex v, of the neighbors of v. A vertex can be represented by an integer $0 \ldots n-1$, where $n = n(G)$ is the order of the graph.

A plane graph will be specified via a *rotation scheme*: a circular ordering, for each vertex v, of the edges incident with v, in the order they appear around v in the plane embedding; this completely specifies the combinatorial embedding of the graph. Rotation schemes are convenient when we represent a graph using adjacency lists; we simply order the adjacency list for each vertex v in clockwise order around v; no additional data structures are required.

We will assume that both forward and backward iteration in adjacency lists is constant time. In terms of plane graphs, given the entry for an edge vu in the rotation scheme ordered adjacency list for a vertex v, we assume that it is a constant time operation to retrieve the entry corresponding to the edge immediately clockwise or counter-clockwise from uv around v. The available C implementation represents each adjacency lists as a linear slice of entries, but a doubly linked list would also suffice [3].

A plane graph is *triangulated* if every face is a 3-cycle, and *weakly triangulated* if every face other than the outer face is a 3-cycle. A graph G is *connected* if given any $u, v \in G$, there exists a u, v-path in G. We say that G is *n-connected* if removing any $n-1$ vertices results in a connected graph.

The input for each algorithm will be a 2-connected weakly triangulated plane graph with n vertices and m edges, represented via adjacency lists. The input size will be n, the number of vertices. Note that $\mathcal{O}(m) = \mathcal{O}(n)$, so it is equivalent to take the input size to be m, the number of edges. Moreover, arbitrary simple planar graphs may be plane embedded and triangulated in $\mathcal{O}(n)$ time, see Boyer and Myrvold [1].

In Section 4, given an edge uv, we will need a constant time operation to find v's entry in u's adjacency list from u's entry in v's list. We define an *augmented adjacency list* to be an adjacency list such that for every edge uv a reference to v's entry in u's list is stored in u's entry in v's list, and vice versa. Given an adjacency list representation of a graph, an augmented adjacency list representation may be constructed in $\mathcal{O}(m)$ time via the following procedure.

Algorithm 2.1.
Input: An adjacency list representation Adj of a graph G.

Output: An augmented adjacency list representation AugAdj of G with the same rotation scheme as Adj.

Step 1: Construct an augmented adjacency list AugAdj with the same rotation scheme as Adj leaving the reference portion of each entry uninitialized.

Step 2: For each vertex v construct an array Wrk[v] of vertex-reference pairs with length $\deg(v)$. For each v from 0 to $n-1$ iterate through AugAdj[v]. For each neighbor u in AugAdj[v] append the pair $(v, r_v(u))$ to Wrk[u], where $r_v(u)$ is a reference to u's entry in AugAdj[v].

Step 3: For each v from $n-1$ to 0 iterate through Wrk[v]. Upon reaching a pair $(u, r_u(v))$ in Wrk[v] the last element of Wrk[u] will be $(v, r_v(u))$; for details on why this must be the case, see the paragraph below. Use $r_u(v)$ and $r_v(u)$ to look up and assign references for the edge uv in AugAdj[u] and AugAdj[v]. Remove $(v, r_v(u))$ from the back of Wrk[u].

After completing Step 2 in Algorithm 2.1 the array Wrk[v] will contain a pair $(u, r_u(v))$ for each neighbor u of v, sorted in increasing order by the neighbor u. Suppose that v is the current vertex at a given iteration of Step 3 in Algorithm 2.1. For each edge $uw \in E(G)$ such that $u < w$ and $v < w$, prior iterations of Step 3 will have initialized the references for uw in AugAdj[u] and AugAdj[w], and also removed the pair $(w, r_w(u))$ from Wrk[u]. Therefore for each $(u, r_u(v))$ in Wrk[v], the array Wrk[u] will contain only entries

for vertices w where $w \leq v$. Since Wrk$[u]$ is sorted in increasing order by the neighboring vertices, the last element of Wrk$[u]$ must be $(v, r_v(u))$.

3 Path Coloring: the Poh Algorithm

In this section we describe a linear time algorithm to path 3-color plane graphs. Let's first recount Poh's path 3-coloring proof strategy. Given a cycle C in a plane graph G we define Int(C) to be the subgraph of G consisting of C and all vertices and edges interior to C. Equivalently, Int(C) is the maximal subgraph of G with outer face C.

Lemma 3.1 (Poh 1990)**.** *Let G be a 2-connected weakly triangulated plane graph with outer face a cycle C. Let $c : V(C) \to S$, $S \subsetneq \{1, 2, 3\}$, be a 2-coloring of C such that each color class induces a nonempty path. There exists an extension of c to a path 3-coloring $c : V(G) \to \{1, 2, 3\}$ such that for all $v \in G - C$, if $vu \in E(G)$ with $u \in C$, then $c(v) \neq c(u)$.*

Proof. If $n(G) = 3$, then $G = C$ and the path 2-coloring of C is a path 3-coloring of G. We proceed by induction on the order of G.

Let P_1, P_2 be the two paths induced by the 2-coloring of the outer face C. Label the vertices of the outer face $C = v_1, v_2, \ldots, v_k$ in clockwise order such that $P_1 = v_1, v_2, \ldots, v_\ell$ and $P_2 = v_{l+1}, v_{l+2}, \ldots, v_k$.

Suppose C is an induced subgraph of G. Let $u, w \in V(G) - V(C)$ be the vertices such that u, v_1, v_k and $w, v_\ell, v_{\ell+1}$ are faces of G; note that u and w are uniquely determined, but it may be that $u = w$. Since C is an induced cycle in G and $G \neq C$, $G - C$ is connected. Let $P_3 = u_1, u_2, \ldots, u_r$ be a u, w-path in $G - C$ of minimal length, and note that P_3 is an induced subgraph of $G - C$.

Color each vertex of P_3 with the color in $\{1, 2, 3\} - S$ not used in the 2-coloring of C. Let $C_1 = v_1, v_2, \ldots, v_\ell, u_r, u_{r-1}, \ldots, u_1$ and $C_2 = u_1, u_2, \ldots, u_r, v_{\ell+1}, v_{l+2}, \ldots, v_k$. The subgraphs Int$(C_1)$ and Int(C_2) together with the coloring c each satisfy the requirements of the lemma. By the inductive hypothesis there exist extensions of c to a path 3-coloring of Int(C_1) and a path 3-coloring of Int(C_2). Since Int(C_1) and Int(C_2) only share the vertices in P_3 on their respective outer faces, the colorings agree and form a path 3-coloring of G.

Suppose C is not an induced subgraph. Then there exists an edge $v_i v_j \in E(G) - E(C)$ such that $i \leq \ell < j$. Let $C_1 = v_1, v_2, \ldots, v_i, v_j, v_{j+1}, \ldots, v_k$ and $C_2 = v_i, v_{i+1}, \ldots, v_j$. By the inductive hypothesis, there exists an extension of c to a path 3-coloring of Int(C_1) and Int(C_2). Again the subgraphs only share vertices on their outer faces, and thus the colorings agree and combine form a path 3-coloring of G. □

Let G be a triangulated plane graph. We may trivially path 2-color the outer triangle. Applying Lemma 3.1 extends this coloring to a path 3-coloring of G.

For an arbitrary plane graph G we may add edges to produce a triangulated plane graph G'. Any path 3-coloring of G' will also be a path 3-coloring of G. Thus Lemma 3.1 proves Theorem 1.1.

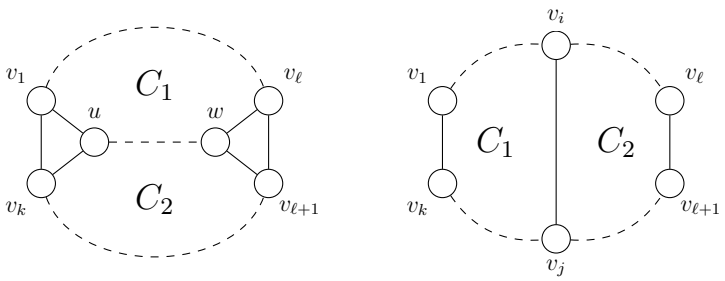

Figure 1: Lemma 3.1 the case C is induced (left) and the case of a chord (right).

In the construction of the path P_3 in Poh's proof, the induced u, w-path was picked to be a u, w-path of shortest length. Thus a natural way to implement Poh's algorithm is to locate u, and then use a breadth-first search to to construct a u, w-path and/or locate a chord edge.

Algorithm 3.2.
Input: Let $C = v_1, v_2, \ldots, v_k$ be a cycle in a 2-connected weakly triangulated plane graph G with adjacency list representation Adj. Let c be an array of colors representing a 2-coloring of C such that each color class induces a path, respectively labelled $P_1 = v_1, v_2, \ldots, v_\ell$ and $P_2 = v_\ell, v_{\ell+1}, \ldots, v_k$. Assume that $c[v] = 0$ for all $v \in \text{Int}(C) - C$.

Output: For each vertex $v \in \text{Int}(C) - C$, a nonzero color will be assigned to $c[v]$ such that c represents a path 3-coloring of $\text{Int}(C)$ extending the original 2-coloring of C. Moreover, if $v \in \text{Int}(C) - C$ has a neighbor $u \in C$, then $c[v] \neq c[u]$.

Procedure: If $k = 3$, then $C = \text{Int}(C)$ and we are done. Otherwise iterate through $\text{Adj}[v_1]$ to locate the vertex u immediately counter-clockwise from v_k. Note that since G is triangulated, v_1, u, v_k is a face of G.

Case 1: Suppose that $c[u] \neq 0$. If $c[u] = c[v_k]$, then $u \in P_2$ and thus $u = v_{k-1}$ since G is triangulated and P_2 is an induced path. Recursively apply the algorithm to the cycle $C' = v_1, v_2, \ldots, v_{k-1}$.

Otherwise $c[u] = c[v_1]$, and it must be that $u = v_2 \in P_1$. Recursively apply the algorithm to the cycle $C' = v_2, v_3, \ldots, v_{k-1}$.

Case 2: Suppose that $c[u] = 0$ and therefore $u \in \text{Int}(C) - C$. Perform a breadth-first search of $\text{Int}(C) - C$ starting from the vertex u. Terminate the search upon locating a vertex $w \notin C$ with adjacent neighbors $v_i \in P_1$ and $v_j \in P_2$ such that $i \neq 1$ or $j \neq k$.

Backtrack along the breadth-first search tree to construct a u, w-path of minumum length $P_3 = u_1, u_2, \ldots, u_r$. Color the vertices in P_3 with the third color not used in the 2-coloring of C. Define

$$C_1 = v_1, v_2, \ldots, v_i, u_r, u_{r-1}, \ldots, u_1 \text{ and } C_2 = u_1, u_2, \ldots, u_r, v_j, v_{j-1}, \ldots, v_k.$$

Apply the algorithm separately to C_1 and C_2. If $i = l$ and $j = \ell + 1$, then C was an induced cycle and we are done. Otherwise, also apply the algorithm to $C_3 = v_i, v_{i+1}, \ldots, v_j$.

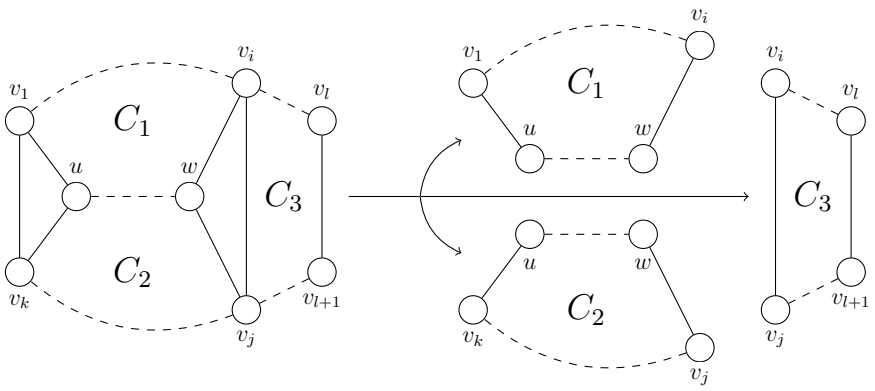

Figure 2: Dividing G along the edge v_iv_j and the uw-path P_3 in Algorithm 3.2.

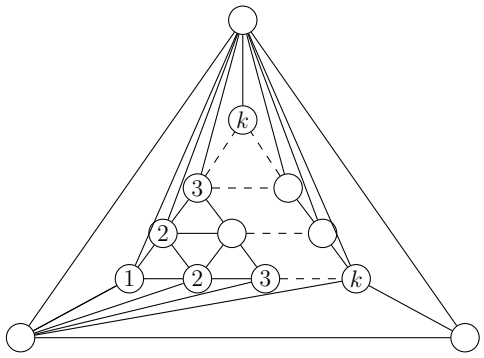

Figure 3: The collection of graphs $\{G_k\}_{k\in\mathbb{N}}$ for which Algorithm 3.2 performs poorly.

Unfortunately Algorithm 3.2 is not linear. Consider the family of graphs $\{G_k\}_{k\in\mathbb{N}}$ depicted in Figure 3. Fix $k \in \mathbb{N}$ and note that $n = n(G_k) = k(k+1)/2 + 3$. Assume that the outer triangle is path 2-colored such that the top vertex is assigned a color distinct from the bottom two. At depth i of Poh's algorithm the shortest path through the interior will be the path of length $r = k - i + 1$ directly along the base of the inner triangle. A breadth-first search of this inner triangle will hit all $r(r+1)/2$ vertices in order to find this path. Therefore the total number of operations performed will be

$$\Theta\left(\sum_{r=1}^{k} \frac{r(r+1)}{2}\right) = \Theta(n^{3/2}).$$

So Poh's algorithm with breadth-first search is $\Omega(n^{3/2})$.

However, the correctness of Poh's algorithm only relies on locating some induced u, w-path. We will show that there exists a linear time implementation of Poh's algorithm so long as we do not always find the shortest possible u, w-path.

For any subgraph H of G, let $N(H)$ be the set of vertices in $V(G)$ with a neighbor in $V(H)$. Our strategy will be to construct an induced path $P_3 = u_1, u_2, \ldots, u_d$ consisting

of vertices in $N(P_1) - V(C)$.

Algorithm 3.3.
Input: Assume that $C = v_1, v_2, \ldots, v_k$ is an induced cycle in a 2-connected weakly triangulated plane graph G with adjacency list representation Adj.

Let c be an array of colors representing a 2-coloring of C such that each color class induces a path, respectively $P_1 = v_1, v_2, \ldots, v_i$ and $P_2 = v_{i+1}, v_{i+2}, \ldots, v_k$. Let c_{P_1} and c_{P_2} be the nonzero colors used on P_1 and P_2, respectively. Assume that $c[v] = 0$ for all $v \in \text{Int}(C) - C$. The vertex v_1 and the entry for the edge v_1v_k in Adj$[v_1]$, together with the coloring c, serve as a representation of C, P_1, and P_2.

Finally, let S be an array of integer marks and let m_{P_1} be an integer such that for each $v \in \text{Int}(C) - C$ we have $S[v] = m_{P_1}$ if and only if $v \in N(P_1)$.

Output: For each vertex $v \in \text{Int}(C) - C$, a nonzero color will be assigned to $c[v]$ such that c represents a path 3-coloring of Int(C) extending the original 2-coloring of C. Moreover, if $v \in N(P_1)$, then $c[v] \neq c_{P_1}$, and if $v \in N(P_2)$, then $c[v] \neq c_{P_2}$.

Step 1: Let u be the neighbor of v_1 immediately counter-clockwise from v_k. If $c[u] \neq 0$ then $u \in C$. But C is an induced cycle, so it must be that $u = v_2$ and $\text{Int}(C) = C = v_1, v_2, v_3$ and Int(C) is already colored. From now on assume that $c[u] = 0$ and $u \notin C$.

Step 2: Let us color an induced path P_3 in $\text{Int}(C) - C$ with the color c_{P_3} distinct from the two colors used on P_1 and P_2. The path P_3 will start at u and end at the unique vertex $w \in \text{Int}(C) - C$ such that w, v_i, v_{i+1} is a face. Note that the vertex w is guaranteed to exist, but in practice we will not know w before constructing the path P_3.

To start, color the vertex u by assigning $c[u] \leftarrow c_{P_3}$. We will now describe the procedure to color and add a vertex to P_3.

Suppose that we have colored an induced path $P_3 = u_1, u_2, \ldots, u_j$ such that $u_1 = u$ and $S[u_\ell] = m_{P_1}$ for each $\ell \in \{1, 2, \ldots, j\}$. Moreover, for each $\ell < j$ assume that no neighbor y of u_ℓ between $u_{\ell-1}$ and $u_{\ell+1}$ counter-clockwise satisfies $S[y] = m_{P_1}$ (define $u_0 = v_k$). Iterate through the neighbors of u_j counter-clockwise from u_{j-1} until we encounter a neighbor y such that $S[y] = m_{P_1}$ or $c[y] = c_{P_1}$.

If $c[y] = c_{P_1}$ then we claim that $y = v_i$ and $u_j = w$. Note that the neighbor x of u_j immediately clockwise from y is also a neighbor of y. Therefore either $c[x] \neq 0$ or $S[x] = m_{P_1}$. But we know that $c[x] \neq c_{P_1}$ and $S[x] \neq m_{P_1}$, so it must be that $c[x] = c_{P_2}$. Since C is an induced cycle, $y \in P_1$, and $x \in P_2$, it must be that $y = v_i$ and $x = v_{i+1}$.

If $S[y] = m_{P_1}$, then we claim that y has not already been added to P_3 and that $u_1, u_2, \ldots, u_j, y$ is an induced path. Since $u_j \in N(P)$, there exists $r \in \{1, 2, \ldots, i\}$ such that u_jv_r is an edge of Int(C). Note that $C_{r,j} = v_1, v_2, \ldots, v_r, u_j, u_{j-1}, \ldots, u_1$ is a cycle in Int(C) and $y \notin \text{Int}(C_{r,j})$. Thus if an edge yu_ℓ exists, it lies between $u_{\ell-1}$ and $u_{\ell+1}$ counter-clockwise in Adj$[u_\ell]$, a contradiction unless $\ell = j$. Therefore we may assign $u_{j+1} \leftarrow y$, color $c[y] \leftarrow c_{P_3}$, and continue constructing the path P_3.

Step 3: So far we have colored an induced path $P_3 = u_1, u_2, \ldots, u_j$ such that

$$C_1 = v_1, v_2, \ldots, v_i, u_j, u_{j-1}, \ldots, u_1 \text{ and } C_2 = u_1, u_2, \ldots, u_j, v_{i+1}, v_{i+2}, \ldots, v_k$$

are cycles. It remains to locate any chords and divide each cycle into induced cycles that we may recursively apply the algorithm to.

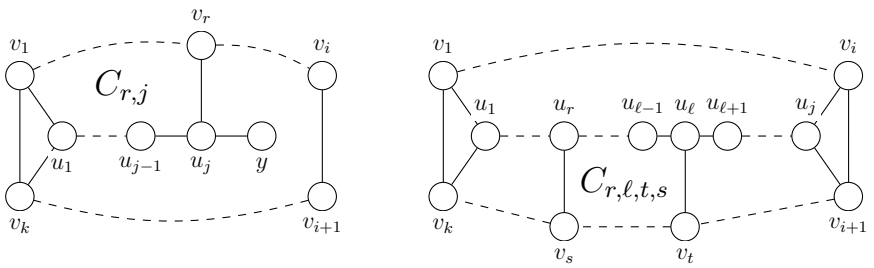

Figure 4: A sketch of the graph structure in Algorithm 3.3 Step 2 (left) and Step 3 (right).

In order to path 3-color $\text{Int}(C_2)$, let us iterate through the vertices of P_3 starting from u_1. We will keep track of an edge $u_r v_s$ by storing u_r and the entry for v_s in $\text{Adj}[u_r]$. Initialize $u_r v_s \leftarrow u_1 v_k$. Additionally, pick m_{P_3} to be a new unique integer mark to identify vertices in $N(P_3)$.

Let u_ℓ be the current vertex and let $u_r v_s$ be the last edge between P_3 and P_2 that we have encountered. Iterate through the neighbors of u_ℓ starting from $u_{\ell-1}$ (v_k when $\ell = 1$) and stopping at $u_{\ell+1}$ (v_{i+1} when $\ell = j$). For each neighbor y of u_ℓ we assign $S[y] = m_{P_3}$. Each time we encounter a neighbor y with $c[y] = c_{P_2}$, then we know that $y = v_t \in P_2$ and $C_{r,\ell,s,t} = u_r, u_{r+1}, \ldots, u_\ell, v_t, v_{t+1}, \ldots, v_t$ is an induced cycle. We have also marked all vertices in $\text{Int}(C_{r,\ell,s,t})$ in $N(P_3)$ with the unique mark m_{P_3}. Make a recursive call to path 3-color $\text{Int}(C_{r,\ell,s,t})$ and assign $u_r v_s \leftarrow u_\ell v_t$.

To path 3-color $\text{Int}(C_1)$ we follow the same procedure used to color $\text{Int}(C_2)$, but iterate through the vertices of P_3 backwards from u_j. At each vertex $u_\ell \in P_3$ iterate through the neighbors in counter-clockwise order from $u_{\ell+1}$ up to $u_{\ell-1}$.

While executing Algorithm 3.3 we iterate through the adjacency list of a vertex a fixed number of times when it is colored and added to a path. Therefore the time complexity of the algorithm is $\mathcal{O}(m) = \mathcal{O}(n)$. See Figure 5 for a concrete example of Algorithm 3.3.

4 Path List Coloring: the Hartman-Škrekovski Algorithm

In this section we describe a linear time algorithm to path color plane graphs such that each vertex receives a color from a specified list. Hartman showed that this is always possible when each vertex is given a list of 3 colors [7, Thm. 4.1]. Around the same time Škrekovski proved a slightly weaker result using the same coloring strategy [9, Thm. 2.2b].

The path list-coloring (or path choosing) procedure discussed in this section is based on the constructive proofs found in Hartman and Škrekovski's papers, but it "localizes" the logic to proceed through the graph one vertex and edge at a time. The resulting algorithm is similar, but will produce different colorings in some situations.

Let C be the outer face of a 2-connected weakly triangulated plane graph G and let $u, v \in C$ be vertices. If $n(G) \geq 3$, then C is a cycle and we define $C(u, v)$ to be the

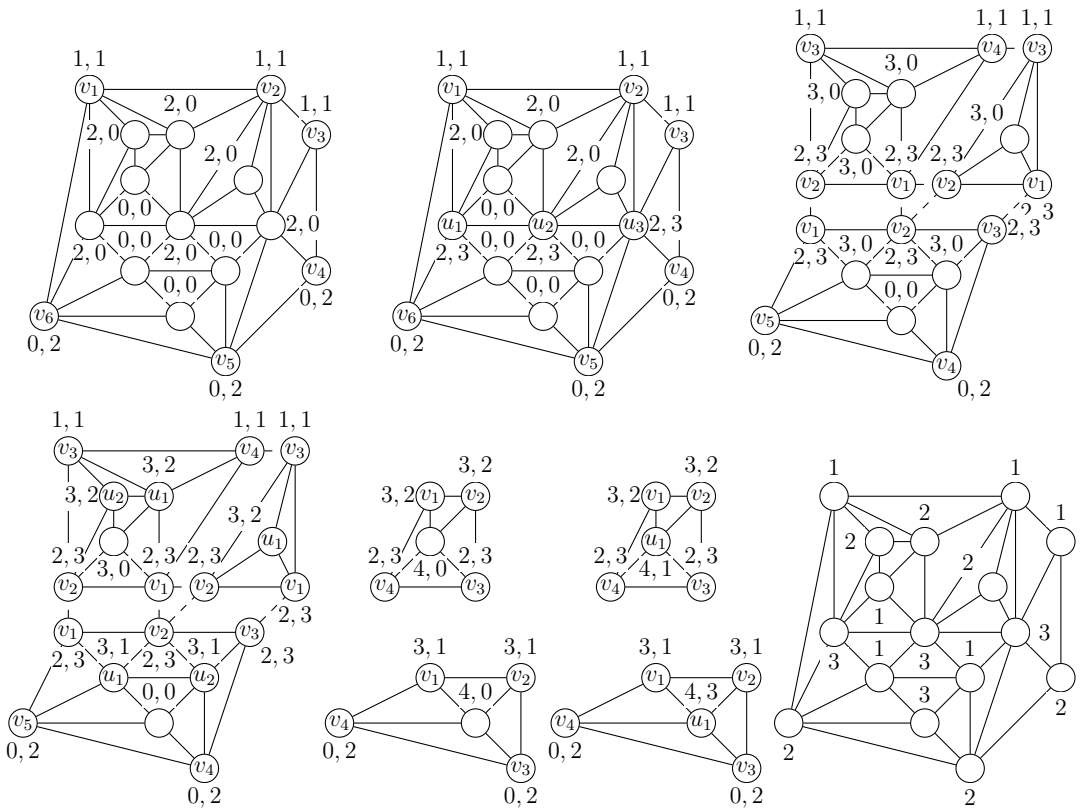

Figure 5: Algorithm 3.3 example. Each vertex $v \in G$ is labelled with $S[v], c[v]$.

clockwise path from u to v around C. If $n(G) < 3$ then we define $C(u, v) = C$ if $u \neq v$, and $C(u, u) = u$.

Given a graph G and a coloring c of G, for each $v \in V(G)$ we define $\deg_c(v)$ to be the number of neighbors of v that share a color with v. Equivalently, $\deg_c(v)$ is the degree of v in the subgraph of G induced by the color class of $c(v)$.

Lemma 4.1. *Let G be a 2-connected weakly triangulated plane graph with outer face C. Let $x, y, z \in C$ be vertices (not necessarily distinct) such that $z \in C(x, y)$. Let $L : V(G) \to P_{<\aleph_0}(\mathbb{N})$ be a function assigning a finite list of colors to each vertex such that*

$$|L(v)| \geq 1 \text{ for } v \in \{x, y, z\}$$
$$|L(v)| \geq 2 \text{ for } v \in C - \{x, y, z\}$$
$$|L(v)| \geq 3 \text{ for } v \in G - C,$$

and if $v \in C(x, z) - z$, then $L(v) \cap L(z) = \emptyset$.

There exists a path coloring $c : V(G) \to \mathbb{N}$ of G such that $c(v) \in L(v)$ for all $v \in G$ and $deg_c(x) \leq 1$, $deg_c(y) \leq 1$, and $deg_c(z) \leq 1$. Moreover, if $v \in G - C(z, y)$ is a neighbor of z, then $c(v) \neq c(z)$.

Proof. Define $c(x)$, $c(y)$, and $c(z)$ to be arbitrary colors in $L(x)$, $L(y)$, and $L(z)$, respectively.

Suppose that $n(G) \leq 2$ and therefore $G = C$. If $n(G) = 1$ then $x = y$ and G is colored. If $n(G) = 2$ and $x \neq y$, then $C = x, y$ and G is colored. If $n(G) = 2$ and $x = y$, then the remaining vertex $v \in G - C(x, y)$ satisfies $|L(v)| \geq 2$. Choose $c(v) \in L(v) - \{c(x)\}$.

We proceed by induction on the order of G. Our strategy will be to apply the inductive hypothesis the blocks of $G - z$. Note that each cut-vertex of $G - z$ must be a neighbor of z in C. Thus each cut-vertex is contained in at most two two blocks, and the block-cut tree of $G - z$ is a path. Label the blocks $H_1, H_2, \ldots, H_{k-1}$ in counter-clockwise order from z around C. Similarly label the neighbors of z in counter-clockwise order as $u_1, \ldots, u_k$. Note that the neighbors $u_2, \ldots, u_{k-1}$ are the cut-vertices of $G - z$; each u_ℓ is contained in $H_{\ell-1}$ and H_ℓ. See Figure 6 for a sketch, and Figure 7 for a concrete example.

Suppose that $u_1 = y$. Since $z \in C(x, y)$, it must be that $x = z$. Define $L'(x) = \{c(x)\}$, $L'(y) = \{c(y)\}$, and $L'(v) = L(v)$ for all $v \in G - x - y$. If $c(x) = c(y)$, then we re-label $x' = y$, $y' = x$, and $z' = x$, and apply the lemma with L', x', y', and z'. Note that x and y will have no neighbors other than each other that share their color. If $c(y) \neq c(x)$, then instead re-label $x' = y$, $y' = x$, and $z' = y$. Note that since $C(z', y') = z'$, we are guaranteed that $\deg_c(z') = 0$. From now on, assume that $u_1 \neq y$.

If $z \neq x$ then let H_i be the block containing x; if $x = u_i$ is a cut-vertex, and is therefore contained two blocks H_i and H_{i+1}, we single out the block H_i. If $z = x$, define $i = 1$.

Similarly, if $z \neq y$ define H_j to be the block containing y; if $y = u_{j+1}$ is a cut-vertex, then pick the block H_j ($j > 0$ since $u_1 \neq y$). If $z = y$ then define $j = k - 1$. In all cases $1 \leq i \leq j < k$.

For any $a, b \in \{1, 2, \ldots, k-1\}$ such that $a \leq b$ define

$$G_{a,b} = G\left[\{z\} \cup V(H_a) \cup V(H_{a+1}) \cup \ldots \cup V(H_b)\right].$$

In other words, $G_{a,b}$ is the induced subgraph of the set $\{z\} \cup \bigcup_{\ell=a}^{b} V(H_\ell)$ on G.

We will path L-choose the blocks of $G - z$ in the order

$$H_i \to H_{i+1} \to \ldots \to H_j \to H_{i-1} \to H_{i-2} \to \ldots \to H_1$$

to produce a path L-choosing of $G_{1,j}$. Finally, if $j < k-1$, we will path L-choose $G_{j+1,k-1}$ as a separate case.

Step 1. To start we will path L-choose $G_{i,i}$.

Case 1.1. Suppose that $i = j$ and therefore $G_{i,i} = G_{i,j}$. If $z = y$ or $c(z) \notin L(u_{i+1})$, define $L_i(v) = L(v) - \{c(z)\}$ for all $v \in N(z) \cap V(H_i)$, and $L_i(v) = L(v)$ for all $v \in V(H_i) - N(z)$. If $z = y$, define $y_i = u_{i+1}$, otherwise $y_i = y$. Similarly, if $x = z$, define $x_i = u_i$, otherwise $x_i = x$. Note that either $y_i = u_{i+1}$ or $|L_i(u_{i+1})| \geq 2$. Define $z_i = x_i$ and apply the inductive hypothesis to path L_i-choose H_i with the designated vertices x_i, y_i, and z_i. The L_i-choosing extends to a path L-choosing of $G_{i,j}$ with $\deg_c(x) \leq 1$, $\deg_c(y) \leq 1$, and $\deg_c(z) = 0$.

Otherwise $z \neq y$ and $c(z) \in L(u_{i+1})$. Define $L_i(u_{i+1}) = \{c(z)\}$, define $L_i(v) = L(v) - \{c(z)\}$ for $v \in N(z) \cap V(H_i - u_{i+1})$, and define $L_i(v) = L(v)$ for all $v \in V(H_i) - N(z)$.

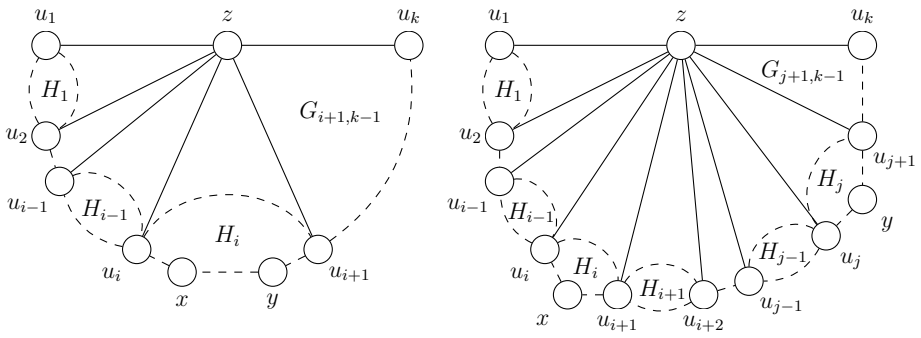

Figure 6: A sketch of the graph structure in the proof of Lemma 4.1 in the case $i = j$ (left) and the case $i \neq j$ (right).

If $x = z$, define $x_i = u_i$, otherwise define $x_i = x$. Define $y_i = y$ and $z_i = u_{i+1}$. Path L_i-choose H_i with x_i, y_i, and z_i. The L_i-choosing of H_i extends to a path L-choosing of $G_{i,j}$ such that $\deg_c(x) \leq 1$, $\deg_c(y) \leq 1$, and $\deg_c(z) = 1$.

Case 1.2. Suppose that $i \neq j$. Define $L_i(v) = L(v) - \{c(z)\}$ for all $v \in N(z) \cap V(H_i)$, and $L_i(v) = L(v)$ for all $v \in V(H_i) - N(z)$. If $z = x$ or if $x = u_i$ is a cut-vertex, then we define $x_i = u_i$, $z_i = u_i$, and $y_i = u_{i+1}$ and path L_i-choose H_i. Note that if $z = x$ then $i = 1$ and $x_1 = u_1$ is the vertex immediately counter-clockwise to z around C.

Otherwise x is not a cut-vertex. Note that $u_i \in C(x, z) - z$ and therefore $|L'(u_i)| \geq 2$. Thus we may define $x_i = x$, $z_i = x$, $y_i = u_{i+1}$ and path L_i-choose H_i.

In both cases the path L_i-choosing of H_i extends to a path L-choosing of $G_{i,i}$ such that $\deg_c(u_{i+1}) \leq 1$, $\deg_c(x) \leq 1$, and $\deg_c(z) = 0$.

Step 2. Suppose that $\ell \in \{i+1, i+2, \ldots, j-1\}$ and we have computed a path L-choosing of $G_{i,\ell-1}$ such that $\deg_c(u_\ell) \leq 1$, $\deg_c(x) \leq 1$, and $\deg_c(z) = 0$.

Define $L_\ell(u_\ell) = \{c(u_\ell)\}$, $L_\ell(v) = L(v) - \{c(z)\}$ for all $v \in N(z) \cap V(H_\ell - u_\ell)$, and $L_\ell(v) = L(v)$ for all $v \in V(H_\ell) - N(z)$. Path L_ℓ-choose H_ℓ with $x_\ell = u_\ell$, $z_\ell = u_\ell$, and $y_\ell = u_{\ell+1}$. We are guaranteed that the path L-choosing of $G_{i,\ell-1}$ and the path L_ℓ-choosing of H_ℓ agree at the shared vertex u_ℓ, and that $\deg_c(u_\ell) \leq 1$ and $\deg_c(u_{\ell+1}) \leq 1$ in H_ℓ. Taken together they form a path L-list coloring of $G_{i,\ell}$ with $\deg_c(u_{\ell+1}) \leq 1$, $\deg_c(x) \leq 1$, and $\deg_c(z) = 0$.

Step 3. Suppose that $i \neq j$ and we have computed a path L-choosing of $G_{i,j-1}$ such that $\deg_c(u_j) \leq 1$, $\deg_c(x) \leq 1$, and $\deg_c(z) = 0$.

If $z = y$, then define $x_j = u_j$, $z_j = u_j$, $y_j = u_{j+1}$, $L_j(v) = L(v) - \{c(z)\}$ for all $v \in N(z) \cap V(H_j)$, and $L_j(v) = L(v)$ for all $v \in V(H_j) - N(z)$. Path L_j-choose H_j and note that it extends to form a path L-choosing of $G_{i,j}$ with $\deg_c(z) = 0$ via the same argument from Step 2.

Otherwise $z \neq y$ and u_{j+1} is the furthest clockwise neighbor of z around C in $C(z, y)$. If $c(z) \in L(u_{j+1})$, then define $L_j(u_{j+1}) = \{c(z)\}$, $L_j(v) = L(v) - \{c(z)\}$ for all $v \in N(z) \cap (H_j - u_{j+1})$, and $L_j(v) = L(v)$ for all $v \in V(H_j) - N(z)$. Path L_j-choose H_j with $x_j = u_j$, $z_j = u_{j+1}$, and $y_j = y$. The path L_j-choosing of H_j agrees with the path L-choosing of $G_{i,j-1}$, and both combine to form a path L-choosing of $G_{i,j}$ such that

$\deg_c(x) \leq 1$, $\deg_c(y) \leq 1$, and $\deg_c(z) = 1$.

If $c(z) \notin L(u_{j+1})$, then define $L_j(v) = L(v) - \{c(z\}$ for all $v \in N(z) \cap V(H_j)$, and $L_j(v) = L(v)$ for all $v \in V(H_j) - N(z)$. Note that $|L_j(u_{j+1})| \geq 2$, thus we may path L_j-choose H_j with $x_j = u_j$, $z_j = u_j$, and $y_j = y$. The L_j-choosing extends to path L-choosing of $G_{i,j}$ with $\deg_c(x) \leq 1$, $\deg_c(y) \leq 1$, and $\deg_c(z) = 0$.

Step 4. We will now color any remaining blocks H_ℓ with $\ell < i$. Suppose that $\ell \in \{1, 2, .., i-1\}$ and that we have computed a path L-choosing of $G_{\ell+1,j}$ such that $\deg_c(z) \leq 1$. Define $L_\ell(u_{\ell+1}) = \{c(u_{\ell+1})\}$, $L_\ell(v) = L(v) - \{c(z)\}$ for all $v \in N(z) \cap V(H_\ell - u_{\ell+1})$, and $L_\ell(v) = L(v)$ for $v \in V(H_\ell) - N(z)$. Path L_ℓ-choose H_ℓ with $x_\ell = u_{\ell+1}$, $y_\ell = u_{\ell+1}$, and $z_\ell = u_{\ell+1}$. Note that $\deg_c(u_{\ell+1}) = 0$ in H_ℓ. Therefore the L_ℓ-choosing extends to a path L-choosing of $G_{\ell,j}$ such that no vertex in $G_{\ell+1,j}$ shares a color with a neighbor in $H_\ell - u_{\ell+1}$.

Step 5. If $j = k - 1$, then the above steps have produced a path L-choosing of G. Otherwise it remains to extend the L-choosing of $G_{1,j}$ to $G_{j+1,k-1}$.

Define $L'(u_{j+1}) = \{c(u_{j+1})\}$, $L'(z) = \{c(z)\}$, and $L'(v) = L(v)$ for $v \in G_{j+1,k-1} - u_{j+1} - z$. Path L'-choose $G' = G_{j+1,k-1}$ with $x' = u_{j+1}$, $z' = u_{j+1}$, and $y' = z$. Let C' be the outer face of G'.

By Step 4 either $\deg_c(z) = 0$ in $G_{1,j}$ or $c(u_{j+1}) = c(z)$ and $\deg_c(z) = 1$ in $G_{1,j}$. Moreover, since $z' = u_{j+1}$, we are guaranteed that the only neighbors of u_{j+1} in G' that share a color with u_{j+1} are in $C'(z', y') = C'(u_{j+1}, z)$. But $C'(u_{j+1}, z) = u_{j+1}, z$, so either $c(z) = c(u_{j+1})$ and $\deg_c(u_{j+1}) = 1$ in G', or $c(z) \neq c(u_{j+1})$ and $\deg_c(u_{j+1}) = 0$ in G'. Because

$$V(G_{1,j}) \cap V(G_{j+1,k-1}) = \{z, u_{j+1}\} = \{x', y'\},$$

it follows that the path L-choosing of $G_{1,j}$ and the path L'-choosing of $G_{j+1,k-1}$ agree and extend to a path L-choosing of G such that $\deg_c(x) \leq 1$, $\deg_c(y) \leq 1$, and $\deg_c(z) \leq 1$. $\square$

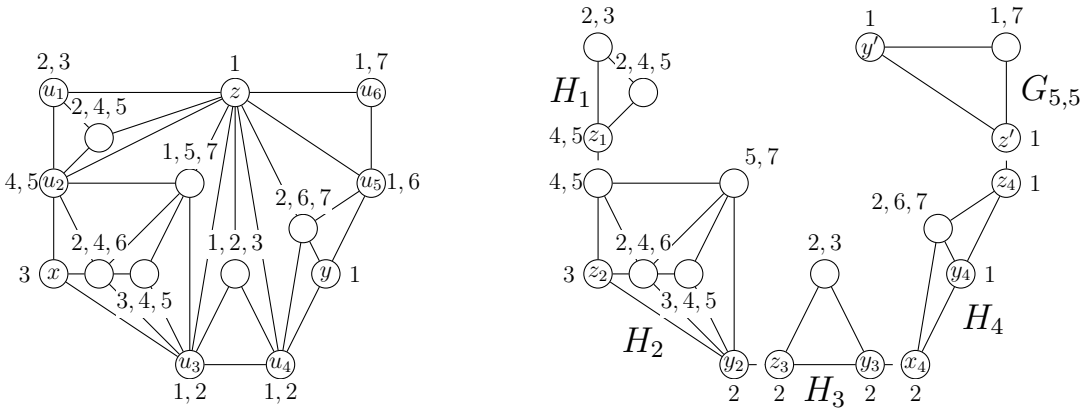

Figure 7: A concrete example step of the path L-choosing precedure in the proof of Lemma 4.1. Each vertex $v \in G$ is labelled with $L(v)$. If $z_\ell = x_\ell$ and/or $z_\ell = y_\ell$, then only z_ℓ is labelled.

A triangulated plane graph G with list assignment L such that $|L(v)| \geq 3$ for all $v \in G$ will satisfy the requirements of Lemma 4.1 with x, y, z chosen arbitrarily on its outer face. Therefore Theorem 1.2 is a special case of Lemma 4.1.

The proof of Lemma 4.1 is a constructive algorithm that may be implemented for plane graphs represented by rotation scheme ordered augmented adjacency lists in $\mathcal{O}(n)$ time. For each vertex $v \in G$ the implementation will track a list $L[v]$ of up to three colors, a integer location mark $S[v]$, and a pair $N[v] = (r_1, r_2)$ of references to neighbor entries in v's augmented adjacency list.

For each $v \in C$, $N[v]$ will contain references to the adjacency list entries for the vertices immediately counter-clockwise and immediately clockwise from v around C. All together, the array of pairs N will provide a representation of C and $\text{Int}(C)$.

We will track the locations of vertices on the outer face by marking each vertex $v \in C(x, z) - z$ such that $S[v] = S[x]$, and marking each vertex $v \in C(z, y) - z$ such that $S[v] = S[y]$. To set up recursive calls we will assign $S[u_\ell]$ a new unique mark, and then for each $v \in C_\ell(u_\ell, u_{\ell+1})$ we will assign $S[v] \leftarrow S[u_\ell]$.

In Step 1, if $z \neq x$ we must instead assign each vertex $v \in C_i(u_i, u_{i+1})$ the mark $S[x]$ so that the marks in $C(x, u_i)$ match to form the continuous segment $C_i(x, u_{i+1})$.

In Step 1, Case 1.1 and Step 3, if $z \neq y$ and $c(z) \notin L(u_{j+1})$, the segment $C_j(x_j, y_j) = C_j(z_j, y_j)$ will need to consist of vertices $v \in C_j(u_j, u_{j+1})$ with $S[v] = S[u_j]$, and vertices $v \in C(u_{j+1}, y)$ with $S[v] = S[y]$. In the recursive call on $H_j = \text{Int}(C_j)$ we will need these distinct marks to compare equal and represent the single face segment $C_j(z_j, y_j)$. To accomplish this without needing to re-mark each vertex, we will track a separate array M initialized such that $M[m] = m$ for each integer mark m. Whenever a mark $S[v]$ is compared with the mark $S[y]$, we will instead compare $M[S[v]]$ with $S[y]$. To join the segment $C_j(u_j, u_{j+1})$ marked with $S[u_j]$ and the segment $C_j(u_{j+1}, y)$ marked with $S[y]$ we simply assign $M[S[u_j]] = S[y]$.

Algorithm 4.2.
Input: Let C be a cycle or length 2-path in a 2-connected weakly triangulated plane graph G represented by augmented adjacency lists. Let $x, y, z \in C$ be vertices such that $z \in C(x, y)$.

Let L be an array of lists of colors such that for $v \in \{x, y, z\}$ the list $L[v]$ has length one, for $v \in C - \{x, y, z\}$ the list $L[v]$ has length two or three, and for $v \in \text{Int}(C) - C$ the list $L[v]$ has length three. Furthermore, for all $v \in C(x, z) - z$ assume that $L[v]$ does not contain the color in $L[z]$.

Let N be an array of pairs of references such that for each $v \in C$ the pair $N[v] = (r_1, r_2)$ contains a reference r_1 to the neighbor the immediately counter-clockwise from v around C, and a reference r_2 to the neighbor immediately clockwise from v around C.

Let S be an array of integers such that for all $v \in C(x, z) - z$ we have $S[v] = S[x]$ and for all vertices $v \in C - C(x, z)$ we have $S[v] \neq S[x]$. Furthermore, for all vertices $v \in C$ we $S[v] \neq 0$ and for all $v \in \text{Int}(C) - C$ we have $S[v] = 0$. Finally, let M be an array of integers such that for all vertices $v \in C(z, y) - z$ we have $M[S[v]] = S[y]$, and for all vertices $v \in C - C(z, y)$ we have $M[S[v]] \neq S[y]$.

Output: Elements will be removed from the lists in L such that for each $v \in \text{Int}(C)$ the list $L[v]$ will contain a single color. The coloring of $\text{Int}(C)$ corresponding to the remaining colors will be a path coloring c such that $\deg_c(x) \leq 1$, $\deg_c(y) \leq 1$, $\deg_c(z) \leq 1$, and if $v \in \text{Int}(C) - C(z, y)$ is a neighbor of z, then $c(v) \neq c(z)$.

Procedure: Let $N[z] = (r_1, r_2)$. Define the vertex u_1 to be the neighbor of z corresponding to r_1, and define v to be the neighbor of z immediately counter-clockwise from u.

Case 1 (Base Case): If $r_1 = r_2$, then $C = z, u_1$ is a length 2 path. If $u_1 \neq x$ and $u_1 \neq y$, remove the remaining color in $L[z]$ from $L[u_1]$. If more than one color still remains in $L[u_1]$, remove arbitrary colors until a single color remains.

Case 2 (Recursive Step): Suppose that $r_1 \neq r_2$. Note that $n(C) > 2$ and therefore C is a cycle.

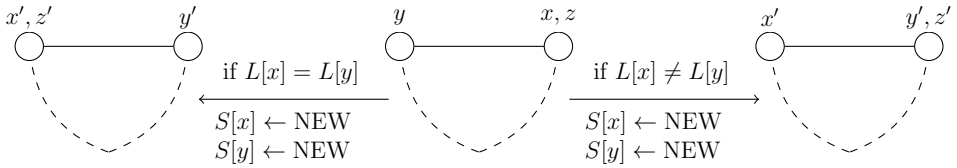

Figure 8: Algorithm 4.2 Case 2.1.

Case 2.1: Suppose that $z = x$ and $u_1 = y$. Assign $S[x]$ and $S[y]$ new unique marks. If $L[x]$ and $L[y]$ contain the same color, make a recursive call re-assigning $x' \leftarrow y$, $y' \leftarrow x$, and $z' \leftarrow x$. Otherwise make a recursive call assigning $x' \leftarrow y$, $y' \leftarrow x$, and $z' \leftarrow y$.

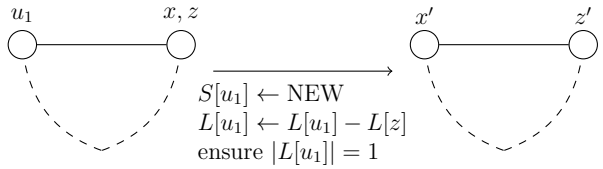

Figure 9: Algorithm 4.2 Case 2.2.

Case 2.2: Suppose that $z = x$ and $u_1 \neq y$. Remove the color in $L[z]$ from $L[u_1]$, and then remove arbitrary colors from $L[u_1]$ until a single color remains. Assign $x' \leftarrow u_1$, $y' \leftarrow y$, $z' \leftarrow z$, set $S[u_1]$ to be a new unique mark, and make a recursive call.

Case 2.3: Suppose that $z \neq x$ and $S[v] = 0$, that is, $v \in \text{Int}(C) - C$. Assign $S[v] = S[x]$, remove the color in $L[z]$ from $L[v]$, and remove the edge zu_1 by adjusting $N[u_1]$ and $N[z]$. Make a recursive call and note that the new "u_1" in this call will be the current vertex v.

Case 2.4: Suppose that $z \neq x$ and $S[v] = S[x]$, in other words $v \in C(x, z)$. Then we remove the edge zu_1 and make two recursive calls. Make the first call with $x_1 \leftarrow x$, $y_1 \leftarrow y$, $z_1 \leftarrow z$, and with $N[v]$ split at the edge vz to only include neighbors clockwise from z. Before the second recursive call we assign $S[v]$ a new unique mark. Make the

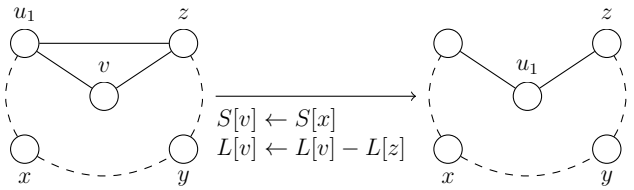

Figure 10: Algorithm 4.2 Case 2.3.

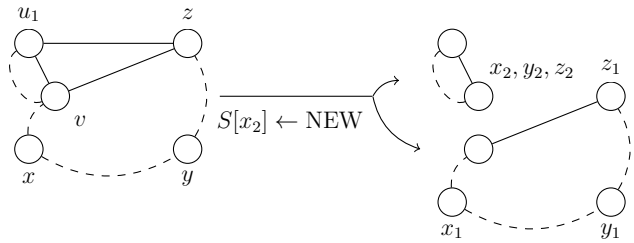

Figure 11: Algorithm 4.2 Case 2.4.

second call with x_2, y_2, z_2 all assigned equal to v and with $N[v]$ split at the edge vz to include only the neighbors of v counterclockwise from z.

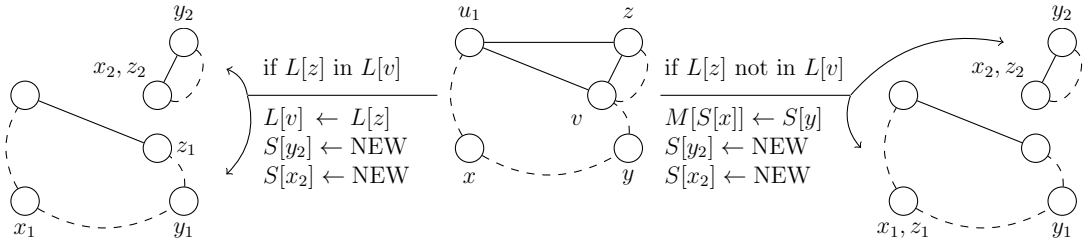

Figure 12: Algorithm 4.2 Case 2.5.1 (left) and Case 2.5.2 (right).

Case 2.5: Suppose that $z \neq x$ and $M[S[v]] = S[y]$, in other words $v \in C(z, y)$. There are two different cases to consider.

Case 2.5.1: Suppose that the color in $L[z]$ is in $L[v]$. In this case we remove all other colors from $L[v]$, remove the edge zu_1, and make two recursive calls. Make the first call with $x_1 \leftarrow x$, $y_1 \leftarrow y$, and $z_1 \leftarrow v$, splitting $N[v]$ at the edge vz to contain only neighbors counter-clockwise from z. Then assign $S[z]$ and $S[v]$ new unique marks, and make a second recursive call with $x_2 \leftarrow v$, $y_2 \leftarrow z$, and $z_2 \leftarrow v$, splitting $N[v]$ at the edge vz to include z and all neighbors clockwise from z.

Case 2.5.2: Suppose that the color in $L[z]$ is not in $L[v]$. Then we remove the edge zu_1, assign $M[S[x]] \leftarrow S[y]$, and make two recursive calls. Make the first call with $x_1 \leftarrow x$, $y_1 \leftarrow y$, and $z_1 \leftarrow x$, splitting $N[v]$ at the edge vz to contain only neighbors counter-clockwise from z. Then assign $S[z]$ and $S[v]$ new unique marks, and make a

second recursive call with $x_1 \leftarrow v$, $y_1 \leftarrow z$, and $z_1 \leftarrow v$, splitting $N[v]$ at the edge vz to include z and all neighbors clockwise from z.

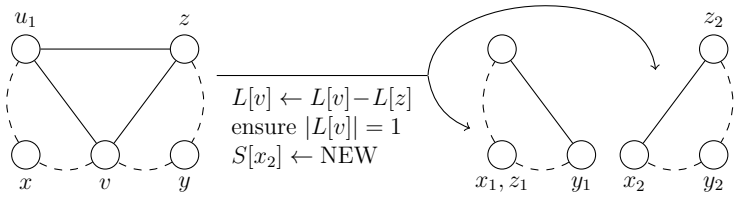

Figure 13: Algorithm 4.2 Case 2.6.

Case 2.6: Suppose that $z \neq x$ and $S[v] \neq 0$, but $S[v] \neq S[x]$ and $M[S[v]] \neq S[y]$, in other words $v \in C(y, x) - x - y$. First remove the color in $L[z]$ from $L[v]$, if it exists, and then remove arbitrary colors until $L[v]$ is of length one. Then remove the edge zu_1 and make two recursive calls. Make the first call with $x_1 \leftarrow x$, $y_1 \leftarrow v$, and $z_1 \leftarrow x$, splitting $N[v]$ at the edge vz to contain only neighbors counter-clockwise from z. Assign $S[v]$ a new unique mark and make the second call with $x_2 \leftarrow v$, $y_2 \leftarrow y$, and $z_2 \leftarrow z$, splitting $N[v]$ at the edge vz to contain z and all neighbors clockwise from z.

Every recursive case in Algorithm 4.2 performs a fixed number of constant time operations. Moreover, each case removes at least one edge from the graph representation, except for Case 2.1 and Case 2.2. In Case 2.1 a single recursive call is made with an input that will not itself satisfy Case 2.1. In Case 2.2 a recursive call is made with an input that will not itself satisfy Case 2.1 or Case 2.2. Therefore the number of operations performed is $\mathcal{O}(m) = \mathcal{O}(n)$. See Figure 14 and Figure 15 for a concrete example of Algorithim 4.2.

5 Experimental performance and parallelism

A C implementation of both algorithms was run on randomly generated triangulated plane graphs up to $10,000,000$ vertices [3], with benchmark timings shown in Figure 16. Benchmark results for a simple breadth-first search algorithm are also shown as a baseline linear algorithm that hits every half-edge of the graph. All benchmarks were run on an x86_64 Intel N100 processor. Timings depicted are the average number of nanoseconds required per graph. The benchmark source code is included with the implementation.

The Poh and Hartman-Škrekovski algorithms color graphs by operating on subgraphs that are disjoint except for vertices on their respective outer faces. Therefore it is possible to maintain a stack of independent recursive frames and have a pool of threads operate on these frames concurrently.

In the case of Poh's algorithm this can be achieved trivially as each frame writes only to vertices interior to the outer face. The algorithm may use a shared lock-guarded stack of frames. If no frames are available, a thread must idle until either a new frame is pushed to the stack by another thread or all threads are idle.

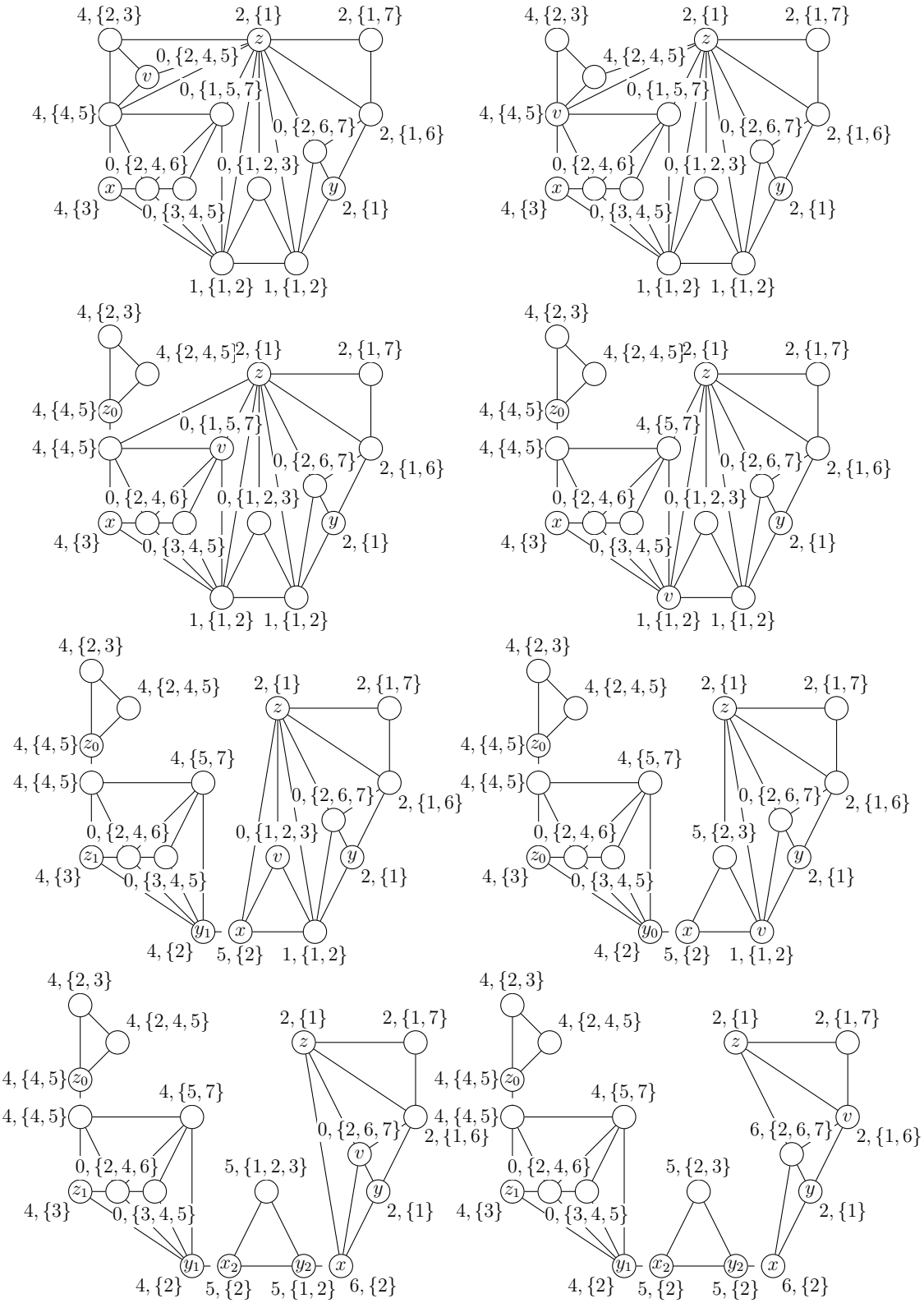

Figure 14: Algorithm 4.2 example. Each vertex $v \in G$ is labelled with $S[v], L[v]$.

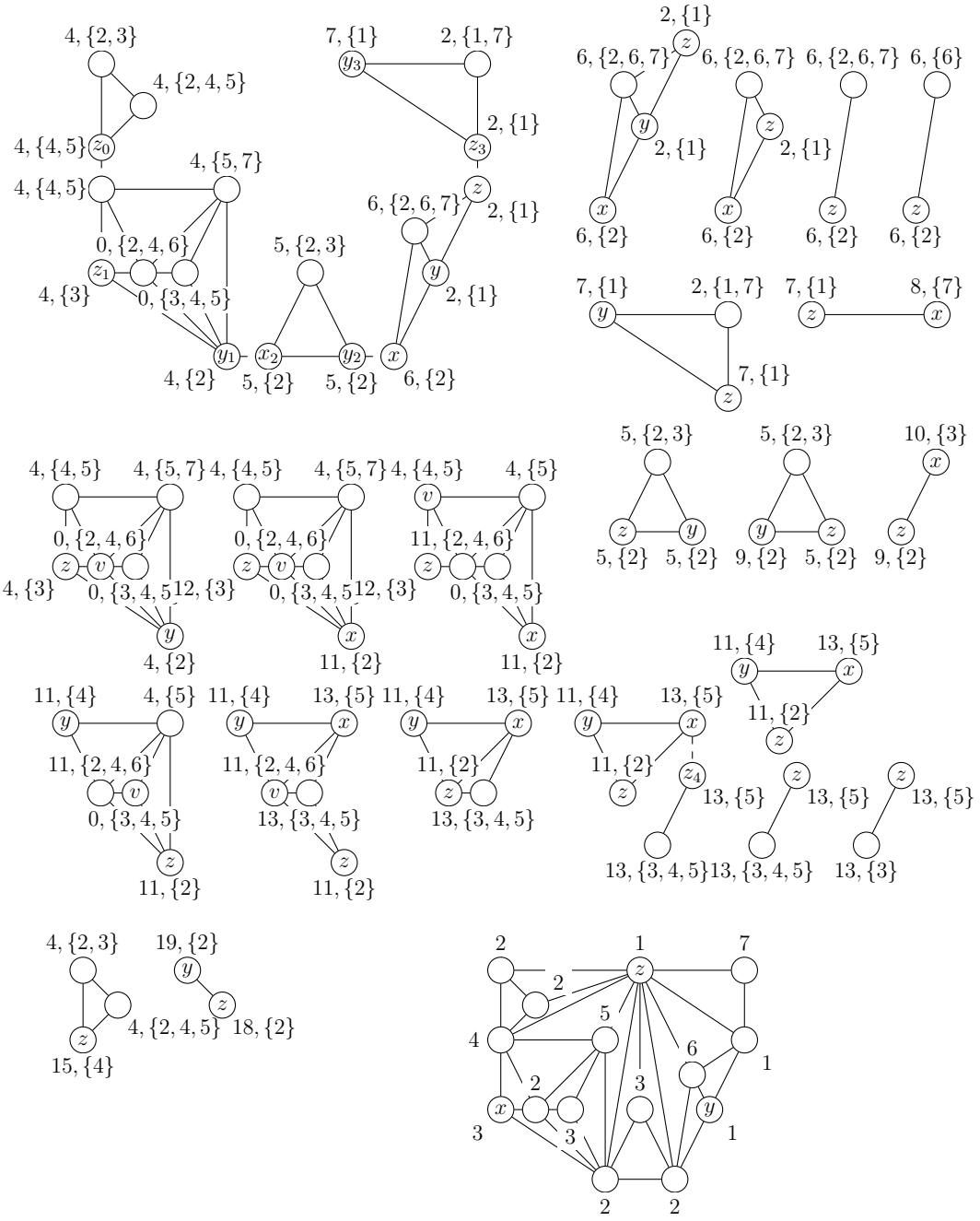

Figure 15: Algorithm 4.2 example continued. Some $n = 2$ recursive calls are not pictured in cases where both vertices are already colored.

	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^7$
Breadth-first search	$2.54 \cdot 10^4$	$3.10 \cdot 10^5$	$3.82 \cdot 10^6$	$1.07 \cdot 10^8$	$1.37 \cdot 10^9$
Poh	$7.16 \cdot 10^4$	$8.25 \cdot 10^5$	$1.01 \cdot 10^7$	$2.17 \cdot 10^8$	$2.70 \cdot 10^9$
Poh (3 threads)	$1.28 \cdot 10^5$	$1.19 \cdot 10^6$	$1.20 \cdot 10^7$	$1.46 \cdot 10^8$	$1.68 \cdot 10^9$
Hartman	$1.03 \cdot 10^5$	$1.25 \cdot 10^6$	$2.93 \cdot 10^7$	$6.40 \cdot 10^8$	$7.60 \cdot 10^9$
Hartman (3 threads)	$2.26 \cdot 10^5$	$2.22 \cdot 10^6$	$3.42 \cdot 10^7$	$5.90 \cdot 10^8$	$6.79 \cdot 10^9$

Figure 16: Benchmarks (ns / graph) for Algorithm 3.3 and Algorithm 4.2.

Benchmarks showed the best performance improvements for Poh's algorithm when using three threads to color plane graphs on the order of 10^6 vertices or more.

The Hartman-Škrekovski algorithm is trickier to adapt for parallel execution. The first difficulty to consider is that certain recursive frames are dependent on each other: the recursive calls made in Algorithm 4.2 cannot be re-ordered, with the exception of those in Case 2.5.1 and Case 2.6. For a concrete example, in Figure 14 the first recursive frame produced cannot start until the vertex z_0 has it's list $L[z_0]$ reduced from $\{4, 5\}$ down to $\{4\}$, which does not happen until much later in Figure 15. A smaller, but no less important detail is that if two threads ever use the same face mark then we risk data races against the array M.

The hurdle of dependent frames can be overcome by maintaining a shared lock-guarded object pool of frames and a stack of frame references. Recursive frames are added to the pool when they appear in the usual course of the algorithm, but are only pushed to the stack when they are ready to be colored. For each vertex $v \in G$ we maintain an optional reference to a frame that will become ready when v has been colored. To allow multiple frames to await the coloring of a single vertex, each frame in the object pool will store an "intrusive linked list," that is, an optional reference to the next frame in line. When a frame is added to the pool and it needs to await a vertex that has another frame waiting on it, the new frame is added to the linked list. Whenever a vertex v is colored we walk the linked list of frames waiting on v and push a reference to each frame onto the shared stack.

We can ensure unique marks accross threads by using a shared atomic mark counter. Contention can be minimized by having threads reserve marks in large chunks, rather than performing an atomic fetch-add operation for each new mark.

Benchmarks for the parallelized Hartman-Škrekovski algorithm showed the best performance benefits when using three threads to color plane graphs on the order of 10^6 vertices or more. The gains observed were less significant than those observed for the Poh algorithm, but that was to be expected as the coordination required between threads is more complicated.

References

[1] J. Boyer and W. Myrvold, On the cutting edge: simplified $O(n)$ planarity by edge addition, *J. Graph Algorithms Appl.* **8** (2004), 241–273.

[2] I. Broere and C. M. Mynhardt, Generalized colorings of outerplanar and planar graphs, *Graph theory with applications to algorithms and computer science* (Kalamazoo, Mich., 1984), pp. 151–161, Wiley-Intersci. Publ., Wiley, New York, 1985.

[3] A. Bross, *Implementing path coloring algorithms on planar graphs*, Masters Project, University of Alaska, 2017, available from `http://github.com/permutationlock/path_coloring_bgl` and `http://github.com/permutationlock/libavengraph`.

[4] G. G. Chappell and C. Hartman, Path choosability of planar graphs, in preparation.

[5] G. Chartrand and H. V. Kronk, The point-arboricity of planar graphs, *J. London. Math. Soc.* **44** (1969), 612–616.

[6] W. Goddard, Acyclic colorings of planar graphs, *Discrete Math.* **91** (1991), no. 1, 91–94.

[7] C. M. Hartman, *Extremal Problems in Graph Theory*, Ph.D. Thesis, University of Illinois, 1997.

[8] K. S. Poh, On the linear vertex-arboricity of a planar graph, *J. Graph Theory* **14** (1990), no. 1, 73–75.

[9] R. Škrekovski, List improper colourings of planar graphs, *Combin. Probab. Comput.* **8** (1999), no. 3, 293–299.

[10] D. B. West, *Introduction to Graph Theory, 2nd ed.*, Prentice Hall, Upper Saddle River, NJ, 2000.