PATH COLORING ALGORITHMS ON PLANE GRAPHS

AVEN BROSS

1. Plane Graphs

A *(simple) graph* $G = (V, E)$ consists of a finite set V of *vertices* and a set E of two element subsets of V known as *edges*. We will often refer to the graph as G and the vertex and edge sets as $V(G)$ and $E(G)$ respectively. As shorthand we will denote an edge $\{u, v\} \in E(G)$ simply as uv. Two vertices $u, v \in V(G)$ are *adjacent* if $uv \in E(G)$. Vertices u and v are known as the *endpoints* of uv. The edge uv is said to be incident to the vertices u and v. The vertices in G adjacent to a vertex v are known as the *neighbors* of v. The *degree* of v is its number of neighbors, denoted $\deg(v)$.

A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $S \subseteq V(G)$ the *induced subgraph* of S on G is the subgraph H defined by $V(H) = S$ and $E(H) = \{uv \in E(G) \mid u, v \in S\}$. We say a subgraph H of a graph G is *induced* if it is the induced subgraph of its vertex set on G. If $v \in V(G)$ we will use $G - v$ to denote the subgraph obtained by removing v and its incident edges from G. Similarly, if H is a subgraph of a graph G, we define $G - H$ to be the subgraph obtained by removing from G all vertices in H and all edges incident to a vertex in H.

A length n path consists of the vertices $v_1, v_2, \ldots, v_n$ and the edges $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n$. A length n cycle, or n-cycle, consists of a length n path and the additional edge v_1v_n. We will often denote a path or cycle G by simply listing its vertices in order, i.e. $G = v_1v_2 \ldots v_n$. If a path $P = v_1v_2 \ldots v_n$ is a subgraph of a graph G, we say P is a *v_1v_n-path* in G. A graph G is *connected* if for every $u, v \in V(G)$, there exists a uv-path in G. If any k vertices may be removed from a graph G with G remaining connected, we say G is *k-connected*.

A *drawing* of a graph maps each vertex to a point in the plane and each edge to a curve connecting its endpoints. A *planar embedding* is a drawing where edge curves intersect only at their endpoints. We say a graph is *planar* if it admits a planar embedding. A planar graph together with a planar embedding is a *plane graph*.

Let G be a plane graph. A *face* of G is a maximal region of the plane not containing any point used in the embedding. The unbounded face is known as the *outer face*. We will always refer to a face by the subgraph of vertices and edges that lie on its border. For brevity, we have not fully formalized curves, regions, or borders. However, the above definitions and results are fairly standard and may be found in many graph theory texts, for example [1].

Theorem 1.1 (Euler's Formula)**.** *If G is a connected plane graph with n vertices, m edges, and f faces, then $n - m + f = 2$.*

A simple corollary of Euler's Formula states that if $n \geq 3$, then $m \leq 3n - 6$. A planar graph is said to be *triangulated* if adding any new edge results in a nonplanar graph. Triangulated plane graphs with $n \geq 3$ vertices have exactly $3n - 6$ edges. A face is said to be a *triangle* if it is a 3-cycle. It is easy to see that all faces in a triangulated plane graph are triangles: if any face has more than three vertices then we may add an edge curve connecting two face vertices without crossing existing edges. Conversely if all faces in a plane graph are triangles, then it is triangulated.

If a plane graph has triangles for all but one face we shall say it is *weakly triangulated*. We will always assume the non-triangle face is the outer face. A 2-connected weakly triangulated plane graph has a cycle for its nontriangle face. Suppose C is a cycle in a 2-connected weakly triangulated plane graph G. Then the subgraph consisting of C and all interior vertices and edges is denoted $\mathrm{Int}(C)$. If $u, v \in V(C)$ then we denote the uv-path in C running clockwise around the cycle with $C[u, v]$. Finally, if $u, v \in V(C)$ we call any edge $uv \in E(G) \setminus E(C)$ a *chord* of the cycle C.

A *rotation scheme* for a graph G is a cyclic ordering of the incident edges of each $v \in V(G)$. Planar embeddings naturally induce a rotation scheme by the counterclockwise order in which edge curves are positioned around each vertex. In fact, with respect to graph algorithms, the induced rotation scheme contains all the useful information of an embedding. Therefore, while we may often visualize plane graphs with drawings, planar embeddings will always be represented solely by their induced rotation scheme.

2. A Brief Review of Coloring Plane Graphs

A *k-coloring* of a graph maps each vertex to one of k possible colors. Equivalently, a k-coloring partitions the vertices of a graph into k disjoint sets called *color classes*. A coloring is *proper* if no pair of adjacent vertices receive the same color, or equivalently, if the color classes each consist of nonadjacent vertices. It is clear not all planar graphs admit a proper 3-coloring since K_4 is planar and requires 4 colors. Whether all planar graphs admit a proper coloring with 4 colors, the Four Color Problem, remained one of the premier open questions in graph theory until it was verified by Appel and Haken in 1976 [7, 8].

An (k, l)-coloring, or a k-coloring with defect l, is a k-coloring such that each vertex has at most l same color neighbors. Generalizations of proper colorings were first introduced by Chartrand et al. in [9], while defective, or improper colorings in particular were introduced simultaneously by Cowen et al. [13], Jones et al. [12], and Jacobson et al. [10]. It was shown in [13] that all planar graphs admit a $(3, 2)$-coloring.

A *path k-coloring*, first introduced in [11], is a k-coloring such that the induced subgraph of each color class consists of one or more disjoint paths. Note that path k-coloring is equivalent to $(k, 2)$-coloring with the added restriction that path coloring forbids cycles. Poh [6] proved that all planar graphs may be path 3-colored, and this proof is easily adapted to an algorithm for coloring plane graphs. Here we provide a naive implementation of Poh's algorithm, running in $\mathcal{O}(n^2)$ time, as well as a more in depth implementation that runs in $\mathcal{O}(n)$.

Let G be a graph and suppose L is a map assigning each vertex $v \in V(G)$ a list of colors. Then a *k-list-coloring* of G, first introduced by Erdös et al. in [14], maps each $v \in V(G)$ to a color in $L(v)$. In [5] Thomassen proved that all planar graphs may be properly 5-list-colored. A planar graph that may not be properly 4-list colored was described by Voigt in [15], so Thomassen's result is best possible.

We may equivalently define the defective (k, l)-list-colorings and path k-list-colorings. Hull and Eaton [4] and Skrekovski [3] independently proved that planar graphs are $(3, 2)$-list-colorable. Hartman [2], also independent, described a procedure for path 3-list-coloring plane graphs. Interestingly, the proofs of Hartman and Skrekovski follow the same coloring algorithm, and thus Skrekovski unknowingly showed the stronger path 3-list-coloring result. Here we present an $\mathcal{O}(n)$ implementation of Hartman and Skrekovski's algorithm.

3. Graph Representations and Time Complexity

The basic operation for all time complexity discussions shall be a single memory reference lookup, integer assignment, or comparison between integers. Memory references are assumed to be integers. We will also treat the allocation of an array as a basic operation, although the operations for initializing its elements are counted separately. In accordance with our assumptions above, removing an element from a linked list or from the back of an array will be considered to be $\mathcal{O}(1)$.

Let G be a plane graph. Vertices will be represented by integers, that is, we shall assume $V(G) = \{1, 2, \ldots, n\}$. We will always denote number of vertices in G with n and the number of edges with m. Note if $n \geq 3$ we have $m \leq 3n - 6$, thus we have $\mathcal{O}(m) = \mathcal{O}(n)$. Vertex properties will be stored in arrays indexed by vertices. Thus accessing or comparing vertex properties shall, in general, be constant time. Finally, colors are assumed to be integers with a coloring of G represented as a vertex property.

For each $v \in V(G)$ we define a linked list called an *adjacency list* containing the neighbors of v ordered according to the rotation scheme of the embedding. The full plane graph G may then represented by a size n array Adj of adjacency lists, indexed by vertices. That is, each $v \in V(G)$ has the adjacency list Adj$[v]$.

We will often wish for the ability to quickly find a neighbor u in v's adjacency list from v's entry in u's list. To allow this lookup in $\mathcal{O}(1)$ time for each $v \in V(G)$ we will instead define Adj$[v]$ to be a linked list of pairs called an *augmented adjacency list*. At the position of u in Adj$[v]$ we will also store a reference to the position of v in Adj$[u]$. An augmented adjacency list representation of a graph G may be constructed from a standard adjacency list representation in $\mathcal{O}(m)$ time via the following algorithm due to Glenn Chappell.

Implementation 3.1. (Augmenting Adjacency Lists)
Input: An adjacency list representation Adj of a graph G.
Output: An augmented adjacency list representation Adj$'$ of G.
Description: We will begin by using Adj to construct an augmented adjacency list representation Adj$'$ of G with the reference portion of each node uninitialized. Next we construct an array Wrk$[v]$ of size $\deg(v)$ for each $v \in V(G)$.

We fill in Wrk as follows. For each v from 1 to n let us walk through Adj$'[v]$. At each neighbor u in Adj$'[v]$ let $r_{v,u}$ be the reference to u's position in Adj$'[v]$ and append the pair $(v, r_{v,u})$ to Wrk$[u]$.

After this process finishes each $u \in V(G)$ will have an array Wrk$[u]$ containing the pairs $(v, r_{v,u})$ for each neighbor v, sorted in ascending by the vertices v. We will now initialize the references of each node of the augmented adjacency lists.

We iterate through the vertices in descending order. Let v be the current vertex. For each $uw \in E(G)$ such that $u < w$ and $v < w$ we shall have initialized the reference for u in Adj$'[w]$ and the reference for w in Adj$'[u]$. We will also have removed the entry $(w, r_{w,u})$ from Wrk$[u]$. It remains to handle edges $uv \in E(G)$ with $v > u$.

For each v from n to 1 we will walk through Wrk$[v]$. For i from 1 to $\deg(v)$ take $(u, r_{u,v}) = \text{Wrk}[v][i]$. Note $u < v$ by our assumptions above. Moreover, Wrk$[u]$ contains no entries for neighbors greater than v so $(v, r_{v,u})$ is the last element of Wrk$[u]$. Thus we may lookup $r_{v,u}$ to find u's node in Adj$'[v]$ and initialize the reference with $r_{u,v}$. We may similarly initialize the reference for v's node in Adj$'[u]$. Finally, we remove $(v, r_{v,u})$ from Wrk$[u]$.

Time Complexity: For each edge $uv \in E(G)$ we make a constant number of assignments to Adj$'$ and Wrk, two reference lookups, and one entry removal from the back of Wrk$[u]$. Therefore the overall complexity of the algorithm is $\mathcal{O}\left(\sum_{k=1}^{m} 1\right) = \mathcal{O}(m)$.

If G is a planar graph without a given embedding we may still construct an adjacency list representation of G, with neighbors simply listed in arbitrary order. There exist numerous algorithms to then simultaneously find an embedding of G and construct an embedding ordered adjacency list representation of the corresponding plane graph in $\mathcal{O}(n)$ time [16, 17, 19, 18]. Moreover, there exist $\mathcal{O}(n)$ algorithms to add edges to the adjacency list representation in order to connect, 2-connect, or triangulate G while maintaining planarity [22, 21, 20]. Thus while algorithms will often assume input graphs are triangulated and plane embedded, arbitrary planar graphs may be modified in linear time to fit these criteria.

Finally, for each vertex v we will also store a pair of references to nodes in Adj$[v]$. Because the rotation scheme is meant to be a cyclic ordering, this will allow us to indicate start and stop nodes for traversing each adjacency list. This will track the "orientation" of each adjacency list around its respective vertex.

4. Path 3-Coloring

In this section we detail two implementations of an algorithm for path 3-coloring plane graphs. We begin by describing the general algorithm proposed by Poh [6].

Algorithm 4.1. (Poh 3-Coloring)

Input: A 2-connected weakly triangulated plane graph G with outer cycle $C = v_1v_2 \ldots v_k$ and a 2-coloring of C such that the color classes induce the paths $P = v_1v_2 \ldots v_l$ and $Q = v_kv_{k-1} \ldots v_{l+1}$.

Output: A path 3-coloring of G such that no vertex in C receives a same color neighbor in $G - C$.

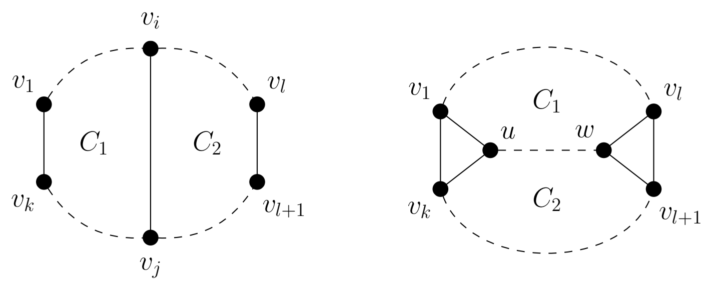

FIGURE 1. The case of a chord (left) and the case no chord exists (right).

Description: If $G - C$ is empty there are no vertices remaining to color. Otherwise the algorithm proceeds as follows.

Suppose there is a chord of C, that is, an edge $v_iv_j \in E(G) \setminus E(C)$ with $i < j$. Since P and Q are induced paths it must be that $v_i \in P$ and $v_j \in Q$. Let C_1 by the cycle consisting of $C[v_j, v_i]$ and the edge v_iv_j, and C_2 the cycle consisting of $C[v_i, v_j]$ and the edge v_iv_j. Observe C_1 and C_2 are each 2-colored such that each color class induces a path. Thus we may apply the algorithm to path 3-color $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$. Since the subgraphs $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$ have only the vertices of the chord v_iv_j in common, the combined coloring forms a path 3-coloring of G.

Suppose no chords of C exist. Let u be the neighbor of v_k immediately clockwise from v_1 and let w be the neighbor of v_l immediately clockwise from v_{l+1}. That is, $u, w \in \mathrm{Int}(C)$ are the unique, but possibly not distinct, vertices such that the cycles uv_1v_k and uv_lv_{l+1} are both triangles.

Since G is weakly triangulated, $G - C$ is nonempty, and C has no chords, $G - C$ is connected. Thus there exists a uw-path in $G - C$. Let T be the shortest such path, and note that therefore T is an induced path. Color T with the remaining color not used on P or Q.

Let C_1 be the cycle consisting of P, T, and the edges v_1u and v_lw. Similarly, let C_2 be the cycle consisting of T, Q, and the edges v_ku and $v_{l+1}w$. Then we may apply the algorithm to path 3-color $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$. Since $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$ have only the vertices of the path T in common, the combined coloring forms a path 3-coloring of G.

Given any plane graph G we may add edges until it is triangulated. Observe that any path coloring of G with the additional edges is also a path coloring of the original G. Therefore by coloring two vertices of the outer triangle with one color and the remaining outer triangle vertex with another, we may apply Poh's algorithm to path 3-color G. This observation yields the following result.

Theorem 4.1 (Poh [6])**.** *All planar graphs are path* 3*-colorable.*

In order to implement Poh's algorithm there are two main obstacles. Firstly, we must have a method to efficiently represent induced paths and subgraphs, as we will be recursively constructing paths and dividing the graph along them. Secondly, we will need an efficient algorithm to locate chords and uw-paths.

In order to efficiently represent induced paths in G we will define a vertex property Mrk[v] for each $v \in V$. To represent an induced path $P = v_1v_2 \dots v_k$ in G we store at the vertices v_1 and v_k as well as an integer I_P. We then assign $\mathrm{Mrk}[v_i] = I_P$ for each $v_i \in V(P)$. Each induced path constructed shall have a unique integer I_P. However, we may subdivide this path, when we split along a chord for example, by assigning different start and end vertices. So two represented paths will have the same mark only if they are segments of a larger induced path and are either disjoint or intersect at a single endpoints.

All paths and cycles discussed as input will be assumed to be subgraphs of a 2-connected weakly triangulated plane graph G which we are working to color. We will now describe our first implementation of Poh's algorithm which uses a breadth first search to find induced paths and chords.

Implementation 4.2. (Poh – Breadth First Search)
Assumptions: Suppose $P = v_1v_2 \dots v_l$ and $Q = v_kv_{k-1} \dots v_{l+1}$ are induced paths such that $C = v_1v_2 \dots v_k$ is a cycle. Finally, assume each path has been colored with a distinct color.
Input: The paths P and Q.
Output: We find an extension of the path 2-coloring of C to a path 3-coloring of $\mathrm{Int}(C)$ such that no vertex in C receives a same color neighbor in $\mathrm{Int}(C) - C$.
Description: Locate the position of v_k in Adj[v_1], generally this will be given as part of the input. Proceeding one vertex further in Adj[v_1] gives us a vertex u such that the cycle uv_1v_k is a triangle. If u is in P, i.e. $u = v_2$, the triangle is colored and we apply the algorithm to the paths $P - u$ and Q. Similarly if w is in Q we apply the algorithm to P and $Q - u$. In either case, if the two remaining paths each consist of a single vertex then there are no remaining uncolored vertices and we terminate the algorithm.

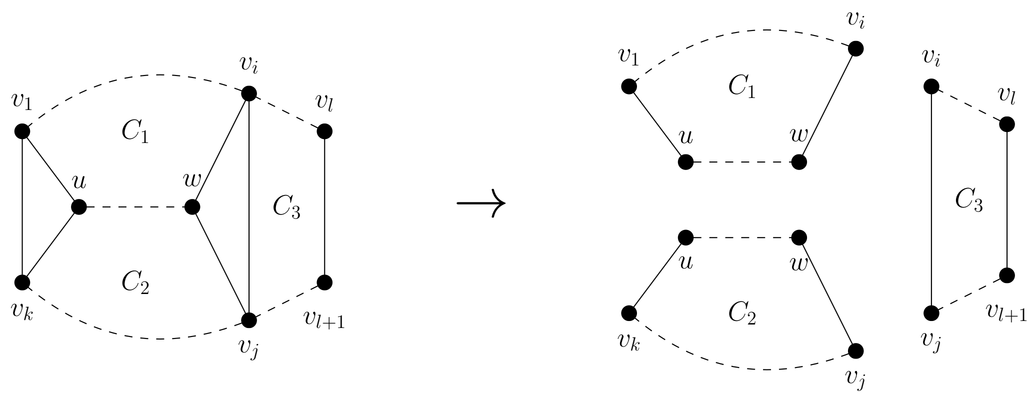

FIGURE 2. Dividing G along the edge v_iv_j and the uw-path.

Otherwise, u is an interior vertex. Perform a breadth first search from u, not not including vertices in C. Terminate the search when we reach a vertex w with neighbors $v_i \in P$ and $v_j \in Q$ such that v_i is immediately past v_j in Adj$[w]$. Such a vertex must exist because Int(C) is weakly triangulated. Backtracking from w along the breadth first search and marking vertices produces an induced uw-path T. We color T with the third remaining color not used on P or Q.

Split P and Q to form the paths $P_1 = v_1v_2 \dots v_i$, $P_2 = v_ip_{i+1} \dots v_l$, $Q_1 = v_kv_{k-1} \dots v_j$, and $Q_2 = v_jq_{j-1} \dots v_{l+1}$. Observe we then have the cycle C_1 consisting of P_1, T, and the edges v_1u and v_iw. Similarly we have the cycle C_2 consisting of T, Q_1, and the edges v_ku and v_jw. We apply the algorithm to P_1 and T to color Int(C_1) and similarly to T and Q_1 to color Int(C_2). If $i = l$ and $j = l + 1$ we are done. Otherwise, we have the cycle C_3 consisting of P_2, Q_2 and the edges p_iq_j and p_kq_l and we apply the algorithm to P_2 and Q_2 to color Int(C_3).

Complexity: In the first step we rotate through Adj$[v_1]$ to find v_k and get an "orientation" within the graph. This orientation must be performed at most once for each vertex, for a total of $\sum_{v=1}^{n} \deg(v) = 2m$ operations.

In the next step we perform a breadth first search from the vertex u. A breadth first search requires at most n lookups. Moreover, the vertex will u will be colored following the search. Thus we perform at most one breadth first search from each vertex, requiring at most n^2 operations. Therefore the complexity of the algorithm is at most $\mathcal{O}(2m + n^2) = \mathcal{O}(n^2)$.

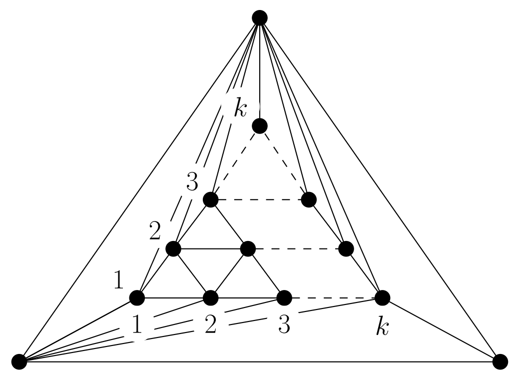

FIGURE 3. The collection of graphs $\{G_k\}$ on which Poh performs poorly.

We define the collection of graphs $\{G_k\}_{k\in\mathbb{N}}$, depicted in 3. Note G_k has $n = \frac{k^2+k}{2}+3$. The number of operations required will be $\mathcal{O}\left(\sum_{i=1}^{k} \frac{k^2+k}{2}\right) = \mathcal{O}(n^{3/2})$. Thus the complexity of the algorithm is at best $\mathcal{O}(n^{3/2})$. In particular, the algorithm is not linear.

Any implementation of Poh's algorithm must find the shortest uv-path within the cycle. Thus Poh's algorithm does not appear to admit a linear time implementation.

However, that the correctness of Poh's algorithm does not require that T be the shortest uw-path, only that T be an induced uw-path. We will show that a linear time implementation exists if we alter Poh's algorithm to instead construct an induced path by walking along the existing path P.

Implementation 4.3. (Poh – Path Trace)
Assumptions: Let $P = v_1v_2 \dots v_l$ and $Q = v_kv_{k-1} \dots v_{l+1}$ be induced paths such that $C = v_1v_2 \dots v_k$ is a chordless cycle, and each path has been colored with a distinct color. In addition, suppose all vertices in $\mathrm{Int}(C) - C$ with a neighbor in P have been marked.
Input: The vertex $u \in \mathrm{Int}(C) - C$ such that the cycle uv_1v_k is a triangle.
Output: We find an induced uw-path T colored with a color distinct from P and Q, where w is the vertex in $\mathrm{Int}(C)$ such that the cycle wv_lv_{l+1} is a triangle. Let C_1 and C_2 be defined as usual by splitting along T. We produce a path 3-coloring of $\mathrm{Int}(C_1)$ such that no vertex in C_1 receives a same color neighbor in $\mathrm{Int}(C_1) - C_1$, and similarly for $\mathrm{Int}(C_2)$.
Description: Initialize T as the path consisting of the single vertex u, coloring u with the designated color. We will recursively add vertices to T until we reach the vertex w.

Suppose we have constructed the induced path $T = t_1t_2 \dots t_d$, with $t_1 = u$. Iterate through $\mathrm{Adj}[t_d]$ beginning from t_{d-1}. If we visit a neighbor v that has a neighbor in P we color v, append T, and repeat this process with v as the new end vertex. If we visit $u \in P$ it must be that $t_d = w$ and the algorithm terminates. Note one of these two cases must occur since t_d has at least one neighbor in P.

We first apply *Face Walk* (4.4) to color $\mathrm{Int}(C_2)$. It remains to color any uncolored vertices in $\mathrm{Int}(C_1)$.

Let $T = t_1t_2 \dots t_d$ be the path constructed above. For each p from 1 to d let us iterate through $\mathrm{Adj}[t_p]$, starting with t_{p+1}. In the case $p = d$ we will define $t_{p+1} = v_l$. Let j be the smallest integer such that $t_{p+1}v_j$ is an edge. Suppose we visit a neighbor $y \in \mathrm{Int}(C_1) - C_1$ followed by a vertex in $v_i \in P$. Note by planarity it must be that $i < j$. We may then apply *Path Trace* (4.3) to color the chordless cycle $C_y = t_pv_iv_{i+1} \dots v_jt_{p+1}$, with the vertex y forming the triangle yt_pv_i. Recall all vertices in $\mathrm{Int}(C_y) - C_y$ with neighbors in P have already been marked.

Implementation 4.4. (Poh – Face Walk)
Input: Paths $P = v_1v_2 \dots v_l$ and $Q = v_kv_{k-1} \dots v_{l+1}$ forming a cycle $C = v_1v_2 \dots v_k$ satisfying the requirements of Poh.
Output: We find an extension of the path coloring of C to a path 3-coloring of $\mathrm{Int}(C)$ such that no vertex in C receives a same color neighbor in $\mathrm{Int}(C) - C$.
Description: If $\mathrm{Int}(C) - C$ is empty there is nothing to color and the algorithm terminates. Otherwise, we proceed as follows.

We will iterate through the vertices of P until we find a chord. All interior vertices visited will be marked to indicate they have a neighbor in P. For each i from 1 to l let us walk through $\mathrm{Adj}[v_i]$ from v_{i-1} to v_{i+1}, excluding v_{i-1} and v_{i+1}. For each neighbor u visited, if $u \notin C$, then $u \in \mathrm{Int}(C) - C$ and we mark it. If $u = v_j \in Q$ then v_iv_j is a chord of C and we stop.

We now split P and Q to form the paths $P_1 = v_1v_2 \dots v_i$, $P_2 = v_ip_{i+1} \dots v_l$, $Q_1 = v_kv_{k-1} \dots v_j$, and $Q_2 = v_jv_{j-1} \dots v_{l+1}$. Let us define C_1 and C_2 as usual. We may then apply *Path Trace* (4.3) to color $\mathrm{Int}(C_1)$, and apply *Face Walk* (4.4) to color $\mathrm{Int}(C_2)$.
Complexity: Note each vertex will be in precisely one colored path. During *Path Trace* (4.3) for each vertex $v \in T$ we will iterate through $\mathrm{Adj}[v]$ at most twice: once to locate the starting neighbor, and once to simultaneously find the next vertex to add to the path and find uncolored vertices above T. During *Face Walk* (4.5) for

each vertex $v \in P$ we iterate through $\mathrm{Adj}[v]$ at most once. Therefore the complexity of the algorithm is $\mathcal{O}\left(\sum_{v=1}^{n} 3 \cdot \deg(v)\right) = \mathcal{O}(6m) = \mathcal{O}(n)$.

5. Path 3-List-Coloring

In this section we describe an implementation of an algorithm for path 3-list-coloring plane graphs. The following general algorithm is due to the independent work of Hartman [2] and Skrekovski [3].

Algorithm 5.1. (Hartman-Skrekovski – Path Color)
Input: Let G be a 2-connected weakly triangulated plane graph with outer cycle $C = v_1v_2 \dots v_k$. Let $x = v_1$ and $y \in C - v_1$. Suppose L is a list assignment for G such that for each vertex $v \in G$

$$\begin{aligned} |L(v)| &\geq 1 \quad \text{if } v = x \text{ or } v = y; \\ |L(v)| &\geq 2 \quad \text{if } v \in C - x - y; \\ |L(v)| &\geq 3 \quad \text{otherwise.} \end{aligned}$$

Output: A path L-list-coloring of G such that x and y each receive at most one same color neighbor.
Description: Select an arbitrary $c \in L(x)$. We will construct an induced path P consisting of vertices of C, that begins at x and proceeds clockwise along the outer face a far as possible towards y. Initialize P to consist of the single vertex x.

Suppose we have constructed an induced path $P = v_{j_1}v_{j_2} \dots v_{j_l}$ with $1 = j_1 < j_2 < \dots < j_l \leq k$. Let us select the largest integer i such that $v_i \in C[v_{j_l}, y]$. If no such i exists we have finished constructing P. Otherwise we append v_i to P and repeat.

Let $P = v_{j_1}v_{j_2} \dots v_{j_l}$ be the path constructed above. For each $i \in \{1, \dots, l-1\}$ if $j_i + 1 < j_{i+1}$ we apply *Remove Path* (5.2) to the cycle formed by $C[v_{j_i}, v_{j_{i+1}}]$ and the edge $v_{j_i}v_{j_{i+1}}$, with designated vertex v_{j_i}.

We finally apply Remove Path (5.2) to the cycle formed by P and $C[v_{j_l}, v_1]$, with designated vertex y.

Algorithm 5.2. (Hartman-Skrekovski – Remove Path)
Input: Let G and $C = v_1v_2 \dots v_k$, x, y, and L all be as in (5.1). Suppose $P = v_1v_2 \dots v_l$ is an induced path such that $V(P) \subseteq V(C[v_1, y])$. Let P be colored with a color c such that $c \in L(v_i)$ for all $i \in \{1, \dots, l\}$. Finally, if $y \notin P$, suppose for every $v \in C[v_{l+1}, y]$, if v has a neighbor in P then $c \notin L(v)$.

Output: A path L-list-coloring of G such that v_k and y each receive at most one same color neighbor, and no vertex in $G - P$ with a neighbor in P receives the color c.

Description: Note G is 2-connected and weakly triangulated. Thus to disconnect G by removing vertices from C we would need to remove vertices $v_i, v_j \in C$ such that $v_i v_j$ is a chord of C. Observe P is an induced path in C, so no vertices in P induce a chord of C. So $G - P$ is connected.

Suppose there is a chord of C with an endpoint in P. Let us select the smallest $i \in \{1, \ldots, l\}$ and largest $j \in \{l + 2, \ldots, k - 1\}$ such that $v_i \in P$ and $v_i v_j$ is a chord of C. Let C_1 be the cycle consisting of $C[v_j, v_i]$ and the edge $v_i v_j$. Similarly, let C_2 be the cycle consisting of $C[v_i, v_j]$ and the edge $v_i v_j$.

We may apply *Remove Path* (5.2) to $\text{Int}(C_1)$ with $P_{C_1} = v_1 v_2 \ldots v_i$, $y_{C_1} = v_i$, and $L_{C_1} = L$. Similarly we may apply (5.2) to $\text{Int}(C_2)$ with $P_{C_2} = v_i v_{i+1} \ldots v_l$, $y_{C_2} = y$, and $L_{C_2} = L$. Note by our assumptions

Suppose there are no chords of C with endpoints in P. Then the only neighbors of P in $C - P$ are the vertices v_k and v_{l+1}. Let L' be a list assignment for $G - P$ defined by

$$L'(v) = \begin{cases} L(v) \setminus \{c\}, \text{if } v \text{ has at least one neighbor in } P; \\ L(v), \text{ otherwise.} \end{cases}$$

Let us define the vertex $x' = v_k$. If $y \in P$ we define $y' = v_{l+1}$, otherwise let $y' = y$. Observe $G - P$ is 2-connected and L' meets the requirements of *Path Color* (5.1) with x' and y'. We may therefore apply (5.1) to path L'-color $G - P$.

In order to implement Hartman and Skrekovski's algorithm there are two main challenges we face. First, we need to be able to remove paths and locate the remaining blocks for recursive calls. Second, we must be able to track the location of vertices on the outer face with respect to the designated vertices x, and y. For example, when constructing a new path P we need to know if the vertex we are looking at lies in $C[x, y]$.

6. Boost Graph Implementation

References

[1] West, D., *Introduction to Graph Theory*, 2nd ed., Pearson, 2001.

[2] Hartman, C., "Extremal Problems in Graph Theory," Ph.D. Thesis, Department of Mathematics University of Illinois at Urbana-Champaign, 1997.

[3] Skrekovski, R., "List Improper Colourings of Planar Graphs," *Combinatorics, Probability, and Computing*, vol. 8, pp. 293-299, 1999.

[4] Eaton, N., Hull, N., "Defective list colorings of planar graphs," *Bulletin of the Institute for Combinatorics and its Applications*, 1999.

[5] Thomassen, C., "Every planar graph is 4-choosable," *Journal of Combinatorial Theory Series B*, 1994.

[6] Poh, K., "On the linear vertex-arboricity of a planar graph," *Journal of Graph Theory*, vol. 14, pp. 73-75, 1990.

[7] Appel, K., Haken, W., "Every planar map is four colorable. Part I: Discharging," *Illinois Journal of Mathematics*, vol. 21, pp. 429-490, 1977.

[8] Appel, K., Haken, W., "Every planar map is four colorable. Part II: Reducibility," *Illinois Journal of Mathematics*, vol. 21, pp. 491-567, 1977.

[9] Chartrand, G., Geller, D. P., Hedetniemi, S., "A generalization of the chromatic number," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 64, Cambridge University Press, 1968.

[10] Andrews, J., Jacobson, M., "On a generalization of chromatic number," *Congressus Numerantium*, vol. 47, pp. 33-48, 1989.

[11] Akiyama, J., Exoo, G., Harary, F., "Covering and packing in graphs IV: Linear arboricity," *Networks*, vol. 11, pp. 69-72, 1981.

[12] Harary, F., Jones, K., "Conditional colorability II: bipartite variations," *Congressus Numerantium*, vol. 50, pp. 205-218, 1985.

[13] Cowen, L., Cowen, R., Woodall, D., "Defective colorings of graphs in surfaces: partitions into subgraphs of bounded valency," *Journal of Graph Theory*, vol. 10, pp. 187-195, 1986.

[14] Erdös, P., Rubin, A., Taylor, H., "Choosability in graphs," *Congressus Numerantium*, vol. 26, pp. 125-157, 1979.

[15] Voigt, M., "List colorings of planar graphs," *Discrete Mathematics*, vol. 120, pp. 215-219, 1993.

[16] Hopcroft, J., Tarjan, E., "Efficient planarity testing," *Journal of the ACM*, vol. 21, pp. 549-568, 1974.

[17] Lempel, A., Even, S., Cederbaum, I., "An algorithm for planarity testing of graphs," *Theory of graphs: International Symposium*, pp.215-232, 1967.

[18] Booth, K., Lueker, C., "Testing for the consecutive ones property interval graphs and graph planarity using PQ-tree algorithms," *Journal of Computer and System Sciences*, vol. 13, pp. 335-379, 1976.

[19] Boyer, J., Myrvold, W., "On the cutting edge: simplified $\mathcal{O}(n)$ planarity by edge addition," *Journal of Graph Algorithms and Applications*, vol. 8, pp. 241-273, 2004.

[20] Eswaran, K., Tarjan, R., "Augmentation problems," *SIAM Journal of Computing*, vol. 5, pp. 653-665, 1976.

[21] Reed, R., "A new method for drawing a graph given the cyclic order of the edges at each vertex," *Congressus Numerantium*, vol. 56, pp.31-44, 1987.

[22] Hagerup, T., Uhrig, C., "Triangulating a planar graph," *Library of Efficient Datatypes and Algorithms*, software package, Max Planck Institute for Informatics, Saarbrücken, 1991.