Implementing Path Coloring Algorithms on Planar Graphs

Aven Bross

June 4, 2017

Abstract

A path coloring of a graph partitions its vertex set into color classes such that each class induces a disjoint union of paths. In this project we consider several algorithms to compute path colorings of graphs embedded in the plane.

We present two implementations of the algorithm to path 3-color plane graphs from Poh's proof in [6]. First we describe a naive implementation that directly follows Poh's procedure. Then we provide an implementation of a modified algorithm that runs in linear time.

The independent work of Hartman [2] and Skrekovski [3] describes an algorithm that takes a plane graph G, as well as a list of 3 colors for each vertex in G, and computes a path coloring of G such that each vertex recieves a color from its list. We provide a linear time implementation of Hartman and Skrekovski's algorithm.

1 Plane Graphs

We will be concerned with simple plane graphs which are, informally, networks consisting of a set of points in the plane, and a set of lines between points such that no lines cross.

Formally, a *graph*, precisely a *simple graph*, is a pair $G = (V, E)$ consisting 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 the vertex and edge sets of a graph G as $V(G)$ and $E(G)$ respectively. As shorthand we will denote an edge $\{u, v\} \in E(G)$ simply as uv. Furthermore, if it is clear by context that v is a vertex, or uv an edge, we will use the notation $v \in G$, or $uv, \in G$.

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 number of neighbors of a vertex v is its *degree*, 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 particular planar embedding is called 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 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 around each vertex. 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 History 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, the complete graph on four vertices, 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 [8, 9].

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 in 1968 by Chartrand et al. in [10]. Defective colorings in particular were introduced about simultaneously around 1985 by Cowen et al. [14], Jones et al. [13], and Jacobson et al. [11]. It was shown in [14] that all planar graphs admit a $(3, 2)$-coloring.

A *path k-coloring* 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. It was conjectured by Broere et. al. [12] that all planar graphs may be path 3-colored. In 1990 Poh [6] and Goddared [7] independatnly proved the conjecture. Planar graphs that do not admit a path 3-coloring were described by Chartrand et. al. [16], and thus the result is best possible.

Poh's proof is constructive and may easily be adapted to an algorithm for path 3-coloring plane graphs. Here we provide a naive implementation of Poh's algorithm, as well as a more in depth implementation that runs in $\mathcal{O}(n)$ time.

Let G be a graph. A *list assignment* for G is a map L assigning each vertex $v \in V(G)$ a list of colors. Given a list assignment L an *L-list-coloring* of G, first introduced by Erdös et al. in [15], maps each $v \in V(G)$ to a color in $L(v)$. We say a graph G is k-choosable if given any list assignment L such that $|L(v)| \geq k$ for all $v \in V(G)$, then G may be properly L-list-colored.

In 1994 Thomassen [5] proved that if G is planar, then G is 5-choosable. A planar graph that is not 4-choosable was described by Voigt [17] in 1993, so Thomassen's result is best possible.

We may equivalently define the properties (k, l)-*choosable* and *path* k-*choosable*. In 1997 Hartman [2] proved that all planar graphs are path 3-choosable. Hartman's result is best possible since path 3-coloring is a special case of path L-list-coloring with lists of size 3. In 1999 Hull and Eaton [4] and Skrekovski [3] independently proved that if G is a planar graph, then G is $(3, 2)$-choosable.

Hartman's proof provides a constructive procedure to construct a path L-list-coloring given a plane graph and an a list assignment with lists of size at least 3. Interestingly, the proofs of Hartman and Skrekovski follow the same coloring algorithm, and thus Skrekovski unknowingly showed the stronger path 3-choosability result. Here we present an $\mathcal{O}(n)$ time 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, inserting or removing an element in a linked list or at the back of an array will be $\mathcal{O}(1)$ time.

Let G be a plane graph. Vertices will be represented by integers, that is, we shall assume $V(G) = \{0, 1, \ldots, n-1\}$. 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 $\mathcal{O}(m) = \mathcal{O}(n)$.

Vertex properties will be stored in arrays indexed by vertices. Thus accessing or comparing vertex properties shall, in general, be constant time.

Colors are assumed to be integers. A coloring of G will thus be represented by an integer 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 $\text{Adj}[v]$.

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 $\text{Adj}[v]$ to be a linked list of pairs called an *augmented adjacency list*. At the position of u in $\text{Adj}[v]$ we will also store a reference to the position of v in $\text{Adj}[u]$.

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.

Algorithm entries, for example *Poh* 3*-Coloring* (4.1), will describe procedures on abstract graphs. Implementation entries, for example *Augment Adjacency Lists* (3.1) below, will describe to procedures working with concrete graph representations. We will provide time complexity analysis for all implementations.

Implementation 3.1. (Augment Adjacency Lists)
Input: An adjacency list representation Adj of a graph G.
Output: An augmented adjacency list representation Adj$'$ of G with the neighbors of each vertex listed in the same order as in Adj.
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 $\mathrm{Wrk}[v]$ of size $\deg(v)$ for each $v \in V(G)$.

We fill in Wrk as follows. For each v from 0 to $n-1$ let us walk through $\mathrm{Adj}'[v]$. At each neighbor u in $\mathrm{Adj}'[v]$ let $r_{v,u}$ be the reference to u's position in $\mathrm{Adj}'[v]$ and append the pair $(v, r_{v,u})$ to $\mathrm{Wrk}[u]$.

After this process finishes each $u \in V(G)$ will have an array $\mathrm{Wrk}[u]$ 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 $\mathrm{Adj}'[w]$ and the reference for w in $\mathrm{Adj}'[u]$. We will also have removed the entry $(w, r_{w,u})$ from $\mathrm{Wrk}[u]$. It remains to handle edges $uv \in E(G)$ with $v > u$.

For each v from $n-1$ to 0 let us walk through $\mathrm{Wrk}[v]$. For i from 1 to $\deg(v)$ take $(u, r_{u,v}) = \mathrm{Wrk}[v][i]$. Note $u < v$ by our assumptions above. Moreover, $\mathrm{Wrk}[u]$ contains no entries for neighbors greater than v so $(v, r_{v,u})$ is the last element of $\mathrm{Wrk}[u]$. Thus we may lookup $r_{v,u}$ to find u's node in $\mathrm{Adj}'[v]$ and initialize the reference with $r_{u,v}$. We may similarly initialize the reference for v's node in $\mathrm{Adj}'[u]$. Finally, we remove $(v, r_{v,u})$ from $\mathrm{Wrk}[u]$.
Time Complexity: For each edge $uv \in E(G)$, $u < v$, we make a constant number of assignments to Adj$'$ and Wrk, two reference lookups, and one entry removal from the back of $\mathrm{Wrk}[u]$. Therefore the overall complexity of the algorithm is $\mathcal{O}(m)$.

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 [18, 19, 21, 20]. 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 [24, 23, 22]. 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.

4 Path Coloring – the Poh Algorithm

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 $C = v_1 v_2 \dots v_k$ and a 2-coloring of C such that the color classes induce the paths $P = v_1 v_2 \dots v_l$ and $Q = v_k v_{k-1} \dots v_{l+1}$.
Output: We find an extension of the 2-coloring of C to a path 3-coloring of G such that no vertex in C receives a same color neighbor in $G - C$.
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 $v_i v_j \in E(G) \setminus E(C)$ 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 $v_i v_j$, and C_2 the cycle consisting of $C[v_i, v_j]$ and the edge $v_i v_j$. 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 $\text{Int}(C_1)$ and $\text{Int}(C_2)$. Since the subgraphs $\text{Int}(C_1)$ and $\text{Int}(C_2)$ have only the vertices of the chord $v_i v_j$ 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, $u, w \in \text{Int}(C)$ are the unique, but possibly not distinct, vertices such that uv_1v_k and uv_lv_{l+1} are each faces.

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 $v_1 u$ and $v_l w$. Similarly, let C_2 be the cycle consisting of T, Q, and the edges $v_k u$ and $v_{l+1} w$. Then we may apply the algorithm to path 3-color $\text{Int}(C_1)$ and $\text{Int}(C_2)$. Since $\text{Int}(C_1)$ and $\text{Int}(C_2)$ 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 path 2-coloring the outer triangle we may apply Poh's algorithm to path 3-color G. This observation yields the following result.

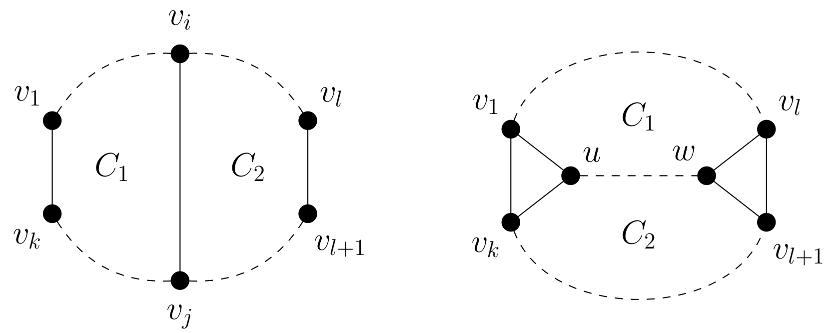

Figure 4.1: The case of a chord (left) and the case no chord exists (right).

Theorem 4.1 (Poh [6] and Goddard [7])**.** *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 colored paths, 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.

To represent induced paths in G we will simply use the color vertex property. Suppose $P = v_1v_2 \dots v_k$ is an induced path in G that has been colored with c_P. Assuming the coloring constructed so far is a path coloring, if $v_i \in P$ then a neighbor u of v_i will have the color c_P if and only if $u \in P$, that is, $u = v_{i-1}$ or $u = v_{i+1}$. Therefore we may represent the entire path by storing just the vertices v_1 and v_k.

Let G be a 2-connected weakly triangulated plane graph with an adjacency list representation. Each call of the algorithm will be provided with a cycle C in G and produce a coloring $\mathrm{Int}(C)$ according to the specification of Poh's algorithm.

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. Additionally, assume each path has been colored with a distinct color.
Input: The paths P and Q, represented by endpoints as described above.
Output: We find an extension of the 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 $\mathrm{Adj}[v_1]$. Proceeding one vertex further in $\mathrm{Adj}[v_1]$ 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.

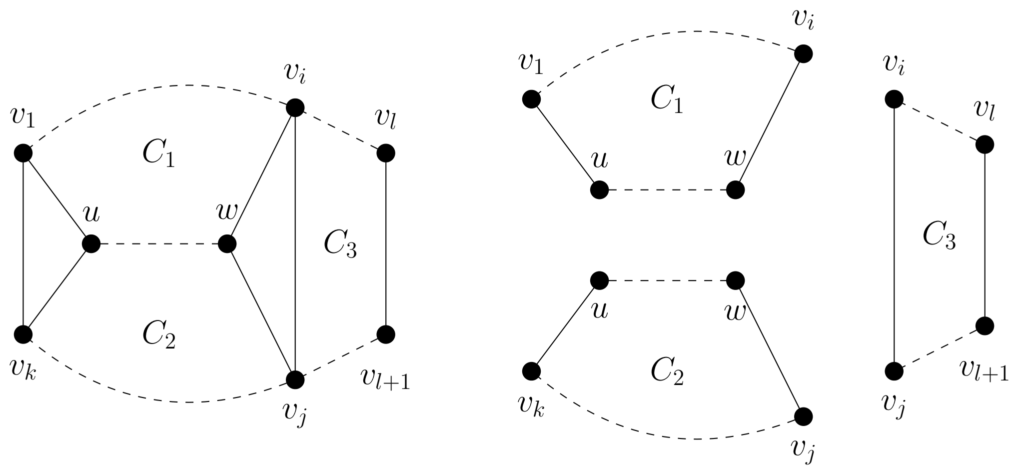

Figure 4.2: Dividing G along the edge v_iv_j and the uw-path.

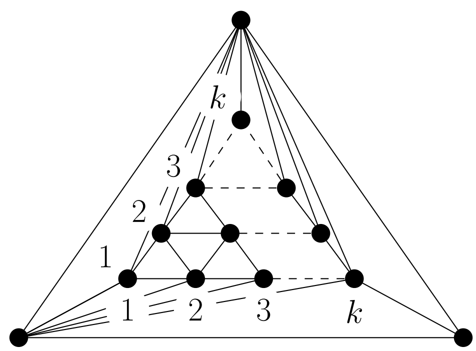

Figure 4.3: The collection of graphs $\{G_k\}$ on which Poh performs poorly.

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 $\mathrm{Adj}[w]$. Such a vertex must exist because $\mathrm{Int}(C) - C$ is connected and $\mathrm{Int}(C)$ is weakly triangulated. Backtracking from w along the breadth first search and coloring vertices produces an induced uw-path T, colored with the remaining color not used on P or Q.

Split P and Q to form the paths $P_1 = v_1v_2 \ldots v_i$, $P_2 = v_ip_{i+1} \ldots v_l$, $Q_1 = v_kv_{k-1} \ldots v_j$, and $Q_2 = v_jq_{j-1} \ldots v_{l+1}$. Observe we have a cycle C_1 consisting of P_1, T, and the edges v_1u and v_iw. Similarly we have a 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 $\mathrm{Int}(C_1)$ and similarly to T and Q_1 to color $\mathrm{Int}(C_2)$. 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 v_iv_j and v_lv_{l+1}, and we may apply the algorithm to color $\mathrm{Int}(C_3)$.

Complexity: In the first step we rotate through $\mathrm{Adj}[v_1]$ 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 $\sum_{v=0}^{n-1} \deg(v) = 2m$ operations.

In the next step we perform a breadth first search from the vertex u. A breadth first search requires at most m 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 nm operations. Therefore the complexity of the algorithm is at worst $\mathcal{O}(2m + nm) = \mathcal{O}(n^2)$.

We define the collection of graphs $\{G_k\}_{k\in\mathbb{N}}$, depicted in Figure 4.3. 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, 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 – 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 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: 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* (4.4) to color $\mathrm{Int}(C_1)$, and apply *Face Walk* (4.3) to color $\mathrm{Int}(C_2)$.
Complexity: See *Path Trace* (4.4).

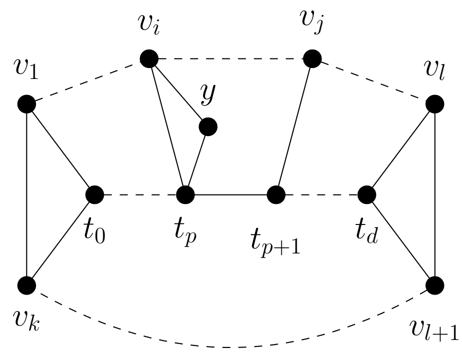

Figure 4.4: Coloring vertices above T in *Path Trace* (4.4), the case v_i is the first neighbor of t_p counterclockwise from t_{p+1} that is in P.

Implementation 4.4. (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, each colored with a distinct color, such that $C = v_1v_2 \dots v_k$ is a chordless cycle. 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 face.
Output: We find an extension of the 2-coloring of C to a path 3-coloring of $\mathrm{Int}(C)$ such that no vertex in C recieves a same color vertex in $\mathrm{Int}(C) - C$.
Description: Initialize T as the path consisting of the single vertex u, coloring u with the remaining color. We will recursively add vertices to T until we reach the unique vertex w such that wv_lv_{l+1} is a face.

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* (4.3) to color $\mathrm{Int}(C_2)$. It remains to color any uncolored vertices in $\mathrm{Int}(C_1)$. Notice that all vertices in $\mathrm{Int}(C_1)$ have neighbors in P. Therefore any uncolored vertex in $\mathrm{Int}(C_1)$ must lie in a path 2-colored chordless cycle of the form $v_iv_{i+1} \dots v_jt_{p+1}t_p$ or $v_iv_{i+1} \dots v_jt_p$. We will use the following procedure to locate all such cycles that contain uncolored vertices and color them using *Path Trace* (4.4).

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 $t_{p+1} = v_l$. Suppose we visit a neighbor $y \in \mathrm{Int}(C_1) - C_1$ followed counterclockwise by a neighbor $v_i \in P$.

Suppose we have previously visited a neighbor of t_p in P, and let $v_j \in P$ be the most recent such neighbor visited. Note by planarity it must be that $i < j$. We may then apply *Path Trace* (4.4) to color the chordless cycle $C_y = v_iv_{i+1} \dots v_jt_p$, with the vertex y forming the face yt_pv_i.

Otherwise none of the neighbors of t_p between t_{p+1} and v_i counterclockwise around t_p were in P. Let j be the smallest integer such that $t_{p+1}v_j$ is an edge, noting again that $i < j$ by planarity. Then the cycle $C_y = t_pv_iv_{i+1}\dots v_jt_{p+1}$ is chordless and we may similarly apply *Path Trace* (4.4) with the vertex y forming the face yt_pv_i.

Complexity: Note each vertex will be in exactly one colored path. Moreover, for a given path P, each vertex $v \in P$ is visited exactly once in each of *Face Walk* (4.3) and *Path Trace* (4.4). During *Path Trace* (4.4) for each vertex $v \in T$ we iterate through $\mathrm{Adj}[v]$ at most twice: once to locate the starting neighbor, and once to find the next vertex to add to the path and find uncolored vertices above T. In *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

$$\mathcal{O}\left(\sum_{v=0}^{n-1} 3 \cdot \deg(v)\right) = \mathcal{O}(6m) = \mathcal{O}(n).$$

5 Path List-Coloring – the Hartman-Skrekovski Algorithm

In this section we describe an implementation of an algorithm for path list-coloring plane graphs with lists of size 3. The following general algorithm is due to the independent work of Hartman [2] and Skrekovski [3]. We will produce an L-list-coloring of the graph by reducing the color list of each vertex to contain a single color. A vertex is therefore considered to be colored when it has a list of size one.

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 $y \in C - v_1$. Suppose L is a list assignment for G such that for each vertex $v \in G$

$$\begin{aligned} |L(v)| &\geq 1 \quad \text{if } v = v_1 \text{ or } v = y; \\ |L(v)| &\geq 2 \quad \text{if } v \in C - v_1 - y; \\ |L(v)| &\geq 3 \quad \text{otherwise.} \end{aligned}$$

We will call v_1 and y *fixed* vertices.
Output: A path L-list-coloring of G such that the fixed vertices v_1 and y each receive at most one same color neighbor.
Description: Select an arbitrary $c \in L(v_1)$. We will construct an induced path P, colored with c, and consisting of vertices from C. The path will begin at v_1 and proceed clockwise along the outer face a far as possible towards y. Initialize P to consist of the single vertex v_1.

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 $v_i \in C[v_{j_l}, y]$ and

$c \in L(v_i)$. If no such i exists we have finished constructing P. Otherwise we append v_i to P and repeat.

For each vertex in $v_i \in P$ let us remove all colors other than c from $L(v_i)$, that is, let us color the path P with c.

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 fixed vertices v_{j_i+1} and $v_{j_{i+1}-1}$.

If $y \in P$ let us define $y' = v_{l+1}$, otherwise $y' = y$. We may finally apply Remove Path (5.2) to the cycle formed by P and $C[v_{j_l}, v_{j_1}]$, with fixed vertices v_k and y'.

Algorithm 5.2. (Hartman-Skrekovski – Remove Path)
Input: Let G and $C = v_1 v_2 \dots v_k$, x, y, and L all be as in (5.1). Suppose $P = v_1 v_2 \dots v_l$ 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 $i \in \{1, \dots, l\}$.

Let G be a 2-connected weakly triangulated plane graph with outer cycle $C = v_1 v_2 \dots v_k$. Let $P = v_1 v_2 \dots v_l$ is an induced path in C. Let $y \in C - P$. Let L be a list assignment for G such that for $v \in G$

$$
\begin{array}{ll}
L(v) = \{c\} & \text{if } v \in P; \\
|L(v)| \geq 1 & \text{if } v = y \text{ or } v = v_k; \\
|L(v)| \geq 2 & \text{if } v \in C - P - y; \\
|L(v)| \geq 3 & \text{otherwise.}
\end{array}
$$

Additionally, assume for every $v \in C[v_{l+1}, y]$, if v has a neighbor in P then $c \notin L(v)$. We will once again refer to v_k and y as the, possibly not distinct, fixed vertices.
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. If $y = v_k$ then y will receive no same colored neighbor in G.
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 $i \in \{1, \dots, l\}$ and largest $j \in \{l+2, \dots, k-1\}$ 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 $v_i v_j$.

Suppose $y \in C[v_{l+2}, v_k]$ and $v_j \in C[v_{l+2}, y]$. We first apply *Remove Path* (5.2) to $\mathrm{Int}(C_1)$ with the colored path $P_1 = v_1 v_2 \dots v_i$, list assignment L, and fixed vertices v_k and y. We then apply *Remove Path* (5.2) to $\mathrm{Int}(C_2)$ with colored path $P_2 = v_i v_{i+1} \dots v_l$, list assignment L, and the singular fixed vertex v_j. The subgraphs $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$ have only the chord $v_i v_j$ in common. The vertex v_i is in the colored path in both $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$, and thus v_i will receive at most one same color neighbor in each. Since v_j is the single fixed vertex in $\mathrm{Int}(C_2)$, v_j will receive no same color neighbors in $\mathrm{Int}(C_2)$. Thus the combined coloring is a path L-list-coloring of G.

Otherwise $v_j \in C[y, v_{k-1}] - y$. Again, we first apply *Remove Path* (5.2) to $\mathrm{Int}(C_1)$ with the colored path $P_1 = v_1 v_2 \dots v_i$, list assignment L, and fixed vertices v_k and v_j. We may then apply *Remove Path* (5.2) to $\mathrm{Int}(C_2)$ with colored path $P_2 = v_i v_{i+1} \dots v_l$, list assignment L, and fixed vertices v_j and y. Observe $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$ have only the chord $v_i v_j$ in common. In both $\mathrm{Int}(C_1)$ and $\mathrm{Int}(C_2)$ we have v_j as a fixed vertex, and v_i is in the colored path. Therefore both will have at most one same color neighbor in each, and the combined coloring is a path L-list-coloring of G.

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

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

Suppose $G - P$ is 2-connected. Recall we assumed that if $v \in C[v_{l+1}, y]$ and v has at least one neighbor in P, then $c \notin L(v)$. Therefore L' meets the requirements of *Path Color* (5.1) if we assign fixed vertices v_k and y.

Suppose $y \neq v_k$. Then we may apply *Path Color* (5.1) to path L'-color $G - P$. By our definition of L' no vertex in $G - P$ with a neighbor in P receives the color P.

Suppose $y = v_k$. If $|L(y)| > 1$ select arbitrary $c_y \in L(y)$ and set $L(y) = \{c_y\}$. Then we may apply *Remove Path* (5.2) to $G - P$ with the colored path y, list assignment L', and the vertices immediately preceeding and subsequent to y on the outer cycle of $G - P$ as fixed vertices. This ensures y receives no same colored neighbors in G.

Finally, if $G - P$ is not 2-connected, then $G - P$ must be a complete graph on one or 2 vertices. It is therefore simple to color $G - P$ such that the requirements hold.

Let G be a plane graph and L a list assignment such that $|L(v)| \geq 3$ for all $v \in G$. We may add edges to G until it is triangulated. Then we may apply *Path Color* (5.1), with arbitrary fixed vertices, to construct an L-list-coloring. This yields the following result.

Theorem 5.1 (Hartman [2]). *All planar graphs are path* 3*-choosable.*

In order to implement Hartman and Skrekovski's algorithm there are two main challenges. First, we need to be able to remove paths and locate the subgraphs for recursive calls. Second, we must be able to track the location of vertices on the outer face with respect to the fixed vertices v_1, and y. For example: when adding a vertex to the path $P = v_{j_1} v_{j_2} \dots v_{j_l}$ in *Path Color* (5.1), we need to know which neighbors of v_{j_l} lie in $C[v_{j_l}, y]$.

For now, let us assume we have solved the second challenge described above. That is, given vertices $u, v, w \in C$, assume we can determine whether $v \in C[u, w]$ in $\mathcal{O}(1)$ time.

Let G be a 2-connected weakly triangulated plane graph with an augmented adjacency list representation. Just as in our implementation of Poh, each call of the algorithm will be provided with a cycle $C = v_1 v_2 \dots v_k$ in G. The job of a particular recursive call is then to color the subgraph $\mathrm{Int}(C)$ such that the requirements of the Hartman-Skrekovski algorithm hold.

We will provide each vertex in G with a boolean vertex property to represent its state. All vertices in C will be have a state indicating they are on the outer face, and likewise interior vertices will have a state indicating they are not in C.

The list assignment L will be represented by vertex property storing a linked list of colors $L[v]$ for each $v \in G$. We will denote the number of colors in the linked list by $|L[v]|$. Once again, we will consider a vertex v colored if $|L[v]| = 1$.

For each vertex $v_i \in C$ we will store a vertex property $\mathrm{Nbr}[v_i]$ called a neighbor range. The neighbor range of v_i will contain a pair of references to nodes in $\mathrm{Adj}[v_i]$. The first reference will point to the position of v_{i-1} in $\mathrm{Adj}[v_i]$ and the second to v_{i+1}. This will provide immediate access to the preceeding and subsequent vertices of v_i in C, and also give nodes in $\mathrm{Adj}[v_i]$ bounding the set of neighbors contained in $\mathrm{Int}(C)$.

Implementation 5.3. (Hartman-Skrekovski – Path Color)
Assumptions: Suppose $C = v_1 v_2 \dots v_k$ is a cycle and $y \in C - v_1$. Assume for each $v \in \mathrm{Int}(C)$

$$\begin{array}{ll} |L[v]| \geq 1 & \text{if } v = v_1 \text{ or } v = y; \\ |L[v]| \geq 2 & \text{if } v \in C - v_1 - y; \\ |L[v]| \geq 3 & \text{otherwise.} \end{array}$$

We again refer to v_1 and y as fixed vertices. Finally, assume for each $v_i \in C$ we have constructed $\mathrm{Nbr}[v_i]$ as described above.
Input: The fixed vertices v_1 and y.
Output: A path L-list-coloring of $\mathrm{Int}(C)$ such that the fixed vertices v_1 and y each receive at most one same color neighbor.
Description:
Complexity: See *Remove Path* (5.4).

Implementation 5.4. (Hartman-Skrekovski – Remove Path)
Assumptions: Suppose $C = v_1 v_2 \dots v_k$ is a cycle and $y \in C - v_1$.

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] Goddard, W., "Acyclic colorings of planar graphs," *Discrete Math*, vol. 91, pp. 91-94, 1991.

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

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

[10] 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.

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

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

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

[14] 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.

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

[16] Chartrand, G., Kronk, H. V., "The point-arboricity of planar graphs," *J. London. Math. Soc.*, vol. 44, pp. 612-616, 1969.

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

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

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

[20] 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.

[21] 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.

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

[23] 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.

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

A Code

A.1 Poh with breadth first search (4.2)

A.2 Poh with face tracing (4.3, 4.4)

A.3 Hartman-Skrekovski (5.3, 5.4)

A.4 Embedding helper functions

A.5 Disjoint set structure