Computing Replacement Paths in Surface Embedded Graphs*

Jeff Erickson
Department of Computer Science
University of Illinois at Urbana-Champaign
jeffe@cs.uiuc.edu

Amir Nayyeri
Department of Computer Science
University of Illinois at Urbana-Champaign
nayyeri2@cs.uiuc.edu

Abstract

Let s and t be vertices in a directed graph G with non-negative edge weights. The replacement paths problem asks us to compute, for each edge e in G, the length of the shortest path from s to t that does not traverse e. We describe an algorithm that solves the replacement paths problem for directed graphs embedded on a surface of any genus g in $O(gn \log n)$ time, generalizing a recent $O(n \log n)$-time algorithm of Wulff-Nilsen for planar graphs [SODA 2010].

1 Introduction

Finding shortest paths in graphs is a fundamental algorithmic problem with an enormous range of applications. In many of these applications, connections represented by edges of the graph occasionally fail, in which case alternative shortest paths that avoid the failed edges must be computed quickly. The *replacement paths problem*, independently posed by Malik *et al.* [20] and Nisan and Ronen [24], formalizes the search for alternative paths as follows. Given a directed graph G with non-negative edge weights and two vertices s and t, the replacement paths problem asks us to compute, for each edge e in G, the length of the shortest (s,t)-path that avoids e. In fact, it suffices to compute replacement paths only for edges in the shortest path in G from s to t; removing any other edge leaves the shortest path unchanged. The replacement paths problem has a long history of algorithms and applications, which is beyond the scope of this paper; we refer the interested reader to Emek *et al.* [6] and Roditty and Zwick [25] for extensive surveys.

Replacement paths in undirected graphs can be computed almost as quickly as a single shortest path. Malik *et al.* [20] describe an algorithm to solve the replacement path problem in $O(m+n \log n)$ time, the same time required to run Dijkstra's algorithm once; a flaw in their algorithm was corrected by Bar-Noy *et al.* [2]. A similar algorithm with the same running time was later independently developed by Hershberger and Suri [14]. Nardelli *et al.* [23] describe a faster algorithm that runs in $O(m\alpha(m,n))$ time on an integer RAM.

The problem appears to be much more difficult in directed graphs. The naive solution is to run Dijkstra's algorithm once for each edge in the original shortest path, which requires $O(mn+n^2 \log n)$ time. Surprisingly, this is nearly the best algorithm known; the only improvement so far is a recent algorithm of Gotthilf and Lewenstein that runs in $O(mn+n^2 \log\log n)$ time [11]. Hershberger *et al.* [15] prove a lower bound of $\Omega(m\sqrt{n})$ time (when $m = O(n^{3/2})$) in the path comparison model proposed by Karger *et al.* [16]. For the special case of *unweighted* directed graphs, Roditty and Zwick [25] describe a randomized algorithm that runs in $O(m\sqrt{n}\log^2 n)$ expected time.

Several previous papers have focused on the natural special case of planar graphs. Bhosle [3] described an algorithm to compute replacement paths in *undirected* planar graphs in $O(n)$ time, again matching the time required to compute a single shortest path [13]. The first breakthrough for directed planar graphs was an algorithm of Emek *et al.* [6] that runs in $O(n \log^3 n)$ time. Klein, Mozes, and Weimann [18, 29] improved the running time to $O(n \log^2 n)$ by replacing a certain divide-and-conquer step with the SMAWK matrix-searching algorithm [1]. Most recently, Wulff-Nilson [30] further improved the running time to $O(n \log n)$; this is the fastest algorithm known to date.

All three near-linear-time algorithms for directed planar graphs rely on a seminal algorithm of Klein for the *multiple-source shortest path* problem in planar graphs [19]. Given a directed plane graph G and a face f of G, Klein's algorithm builds a data structure of size $O(n)$, in $O(n \log n)$ time, such that the shortest path distance between any vertex incident to f and any other vertex of G can be retrieved in $O(\log n)$ time. Wulff-Nilsen's algorithm [30] applies the underlying data structures and analysis of Klein's algorithm directly; the other two algorithms [6, 18, 29] invoke Klein's algorithm as a subroutine.

Cabello and Chambers [4] generalized Klein's algorithm to graphs embedded on arbitrary surfaces, solving

*This research was partially supported by NSF grant CCF 09-15519. See http://www.cs.uiuc.edu/~jeffe/pubs/repath.html for the most recent version of this paper.

the multiple-source shortest path problem in $O(g^2 n \log n)$ time and $O(n)$ space, where g is the genus of the underlying surface. This time bound was recently improved to $O(gn \log n)$ by Cabello *et al.* [5]. Cabello and Chambers' key insight is to interpret the multiple-source shortest path problem as a *parametric shortest path* problem. Their algorithm starts with a shortest path tree T rooted at an arbitrary vertex of f, and then maintains T as the root moves *continuously* around the boundary of f. (See Section 2.3 for more details.)

In this paper, we show that the parametric shortest-path infrastructure developed by Cabello *et al.* [4, 5] can be used to solve the replacement paths problem in $O(gn \log n)$ time when the input graph is embedded on an orientable surface of genus $g \geq 0$. To the best of our knowledge, this is the first replacement-paths algorithm for directed surface graphs that runs in subquadratic time. For the special case of directed planar graphs, our algorithm has the same $O(n \log n)$ running time as Wulff-Nilsen's algorithm [30], but we believe both our algorithm and its analysis are simpler. Like earlier algorithms [6, 30], our algorithm computes only the *lengths* of the replacement paths, but it can be modified easily to return the actual replacement paths in $O(1)$ additional time per edge.

After reviewing some background in Section 2, we reformulate the replacement path problem in *arbitrary* directed graphs in terms of parametric shortest paths in Section 3. We use this generic algorithm as a template for our algorithm for surface graphs in Section 4.

2 Preliminaries

2.1 Graphs and (Shortest) Paths

An edge in a directed graph G is an ordered pair of vertices; we use the mnemonic notation $u \to v$ to denote the directed edge from vertex u to vertex v.

A directed *walk* in G is a sequence of vertices $v_0 \to v_1 \to \cdots \to v_k$ such that each adjacent pair $v_{i-1} \to v_i$ is an edge in G. A walk with distinct vertices is called a *path*. We say that a path $P = v_0 \to v_1 \to \cdots \to v_k$ *traverses* each edge $v_{i-1} \to v_i$ and *avoids* every other edge in G. For any indices $i < j$, we let $P[i..j]$ denote the *subpath* $v_i \to v_{i+1} \to \cdots \to v_j$ from the ith vertex of P to the jth vertex of P. A *prefix* of P is a subpath that contains the first vertex of P; a *suffix* of P is a subpath that contains the last vertex of P. The *concatenation* $P \cdot Q$ of two paths $P = v_0 \to v_1 \to \cdots \to v_k$ and $Q = v_k \to v_{k+1} \to \cdots \to v_l$ is the walk $v_0 \to v_1 \to \cdots \to v_k \to v_{k+1} \to \cdots \to v_l$.

To simplify our presentation and analysis, we assume that in each of the graphs we consider, every pair of vertices is connected by a unique shortest path; in particular, we assume that replacement paths are unique. The Isolation Lemma [22] implies that this assumption can be enforced (at least with high probability) by perturbing the edge weights with random infinitesimal values [8].

2.2 Surfaces

A *surface* (more formally, a *2-manifold*) is a compact Hausdorff space in which every point has an open neighborhood homeomorphic to the plane. A simple cycle in a surface Σ is (the image of) an injective continuous map $\gamma : S^1 \to \Sigma$. A simple cycle γ is *separating* if its removal disconnects the surface. The *genus* of a surface Σ is the maximum number of simple, disjoint cycles in Σ whose deletion leaves the surface connected. Two compact, connected, orientable surfaces are homeomorphic if and only if they have the same genus.

A *path* on a surface Σ is (the image of) a continuous map $\alpha : [0,1] \to \Sigma$. Two paths α and β are *homotopic* if there is a continuous map $h : [0,1] \times [0,1] \to \Sigma$ such that $h(0,\cdot) = \alpha$, $h(1,\cdot) = \beta$, $h(\cdot,0) = \alpha(0) = \beta(0)$, and $h(\cdot,1) = \alpha(1) = \beta(1)$. The function h is called a *homotopy* between α and β.

An *embedding* of a graph G on a surface Σ maps vertices to distinct points and (undirected) edges to interior-disjoint curves. The *faces* of the embedding are maximal connected subsets of Σ that are disjoint from the image of the graph. An embedding is *cellular* if each of its faces is homeomorphic to the plane. Any cellular embedding can be represented combinatorially by a *rotation system*, which specifies the counterclockwise ordering of edges incident to each vertex of G [21]. Two paths in a surface-embedded graph are considered homotopic if their images on the surface are homotopic.

Let $P = v_0 \to v_1 \to \cdots \to v_k$ be a directed path in an embedded graph G. We say that the edge $u \to v_i$ *enters P from the left* (resp. right) if the vertices v_{i-1}, u, and v_{i+1} are ordered clockwise (resp. counterclockwise) around v_i, according to the embedding's rotation system. An edge $v_i \to u$ *leaves P from the left* (resp. right) if its reversal $u \to v_i$ enters P from the left (resp. right). These definitions require that $0 < i < k$ and that u is not a vertex in P; edges incident to v_0 or v_k are considered neither left nor right of P.

2.3 Parametric Shortest Paths

Cabello *et al.* [4, 5] solved the multiple-source shortest path problem in surface-embedded graphs by recasting it in the *parametric shortest path* paradigm introduced by Karp and Orlin [17]. The input to the parametric shortest path problem is a graph whose edge weights are linear functions of a parameter λ; specifically, let

$w_\lambda(e) = w(e) + \lambda \cdot w'(e)$, for given functions $w, w' : E \to \mathbb{R}$. (In Karp and Orlin's original formulation, $w'(e) \in \{0, 1\}$ for every edge e.) Let T_λ denote the shortest-path tree rooted at a fixed source vertex s with respect to the weights w_λ. The goal of the problem is to compute T_λ for *all* λ in a certain range.

To solve this problem, Karp and Orlin [17] propose maintaining T_λ while *continuously* increasing the parameter λ. Although shortest-path distances vary continuously as a function of λ, the combinatorial structure of T_λ changes only at certain *critical values* of λ. At each critical value, an edge $v \to w$ *pivots* into the shortest-path tree, replacing some other edge $u \to w$; in other words, v becomes the new predecessor of w. Karp and Orlin [17] show that by storing T_λ in an appropriate dynamic tree data structure [26, 12, 27], it is possible to update T_λ in $O(\log n)$ time at each critical value; moreover, during the simulation, the current shortest path distance from s to any other vertex can be retrieved in $O(\log n)$ amortized time [17]. The overall running time of Karp and Orlin's algorithm depends on two additional factors: the amortized time needed to identify the next pivot, and the total number of pivots.

Cabello *et al.* [4, 5] develop efficient algorithms for a narrow special case of the parametric shortest path problem, where the input graph is undirected and has a cellular embedding on some surface, and the source s of the shortest-path tree moves continuously along an edge uv from one end to the other. Their algorithm finds pivots by maintaining and querying the complementary dual subgraph $C_\lambda^* = (G \setminus T_\lambda)^*$. When the input graph G is planar, C_λ^* is a spanning tree of the dual graph G^*, which can be maintained in another dynamic tree data structure [26, 28], so that the next pivot can be computed in $O(\log n)$ amortized time. (Klein's multiple-source shortest-path algorithm [19] uses the same primal and dual dynamic tree data structures, but performs a different sequence of pivots.) For surfaces of genus $g > 0$, the dual subgraph C_λ^* is a spanning tree plus $2g$ extra edges. Cabello *et al.* describe a canonical decomposition of C_λ^* into $O(g)$ edge-disjoint subtrees; by maintaining each of these subtrees in its own dynamic tree data structure, the next pivot can be predicted in $O(g \log n)$ amortized time [4, Lemma 3.2].

With only trivial modifications, the algorithm of Cabello *et al.* can be adapted to directed graphs where exactly one edge has non-constant weight, with the following time and space bounds:

Lemma 2.1 (Cabello *et al.* [5]). *Let G be a directed graph with non-negative edge weights, cellularly embedded on an orientable surface of genus g, and let s be a fixed vertex of G. After $O(n \log n)$ preprocessing time, we can maintain a data structure of size $O(n)$ that supports the following operations:*

- UPDATE(e, w)*: Change the weight of the edge e to $w \geq 0$, in $O((g+p)\log n)$ amortized time, where p is the number of pivots that the tree go through as the weight of e gradually changes to w.*[1]
- DISTANCE(v)*: Return the shortest-path distance in G from s to v in $O(\log n)$ amortized time.*

These same operations can be supported in *arbitrary* directed graphs using the parametric shortest-path algorithms of Karp and Orlin [17] and Young *et al.* [31]. The latter algorithm requires $O(m + n \log n)$ time to perform an UPDATE, after which any DISTANCE query can be answered in constant time. However, unlike the specialized algorithm of Cabello *et al.*, these generic algorithms require the entire graph to be preprocessed anew for each UPDATE. Dynamic data structures supporting UPDATE and DISTANCE operations are also described by Klein [19, Section 6] for directed planar graphs, and by Frigioni *et al.* [9, 10] for several families of graphs including graphs of bounded genus. However, at least for our application, these data structures are not as efficient as the structure described in Lemma 2.1.

3 General Graphs

We now describe our general strategy for solving the replacement path problem using parametric shortest path operations. The algorithm we describe can actually be applied to *arbitrary* directed graphs, although its running time then matches the naive algorithm that runs Dijkstra n times.

Fix an input directed graph $G = (V, E)$, a non-negative weight function $w : E \to \mathbb{R}^+$, and two vertices s and t. Without loss of generality, we assume that s has out-degree 1 and in-degree 0, and that t has in-degree 1 and out-degree 0. Let $P = v_0 \to v_1 \to \ldots \to v_k$ denote the shortest path in G from $s = v_0$ to $t = v_k$. For each index i, let e_i denote the edge $v_{i-1} \to v_i$.

Our goal is to compute, for each i, the length of the shortest path P_i from s to t that avoids e_i. (Removing an edge that is not in P does not change the shortest path.) Because P is a shortest path, the replacement path P_i has the form $P[0..Out(i)] \cdot D_i \cdot P[In(i)..k]$ where $Out(i)$ and $In(i)$ are indices such that $0 < Out(i) < i \leq In(i) < k$, and D_i is a path that is disjoint from P except at its endpoints. We call the middle subpath D_i of P_i a *detour*.

Following Emek *et al.* [6], we define a sequence of edge-weighted graphs $G_1, G_2, \ldots, G_k$ such that for each i, the shortest path from s to t corresponds to a

[1] The published version of this result [4] has a weaker amortized time bound of $O(gp \log n)$. Using this version increases the running time of our algorithm to $O(g^2 n \log n)$.

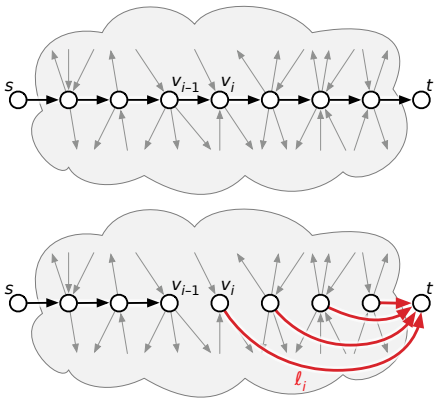

Figure 1. Transforming G (top) into G_i (bottom).

shortest e_i-avoiding path in G. For each index i, let $W_i := \sum_{j=i+1}^{k} w(e_j)$; thus, W_i is the length of the suffix $P[i..k]$. The graph G_i is defined by modifying the graph G, by adding edges $\ell_j := v_j \to t$ with weight W_j, for each index $j \geq i$, and then deleting all edges in the suffix $P[i-1..k]$. See Figure 1. The shortest path from s to t in G_i has the form $P[0..Out(i)] \cdot D_i \cdot \ell_{In(i)}$; the length of this path is the same as the length of P_i. Thus, the replacement path problem boils down to finding the shortest path from s to t in each graph G_i.

Our algorithm computes these distances by maintaining a single graph G^+, constructed from G by adding the edges $\ell_j := v_j \to t$ for all $1 \leq j \leq k$, and changing the weights of its edges. We initially assign each edge ℓ_j infinite weight, so that shortest paths in G^+ are identical to shortest paths in G. Our algorithm computes the shortest path tree T_s rooted at s in G^+, and then preprocesses G^+ and T_s into a data structure that supports the UPDATE and DISTANCE operations described in Lemma 2.1. We implement UPDATE as a parametric shortest path operation; we maintain the shortest-path tree T_s as the weight of edge e changes *continuously and monotonically* from its previous value to its new value.

After preprocessing, our algorithm proceeds in phases indexed from k down to 1. In the ith phase, we UPDATE the weight of e_i to ∞, and then UPDATE the weight ℓ_i to W_i. After the ith phase, the edge weights are consistent with the graph G_i, so DISTANCE(t) is the length of the replacement path P_i. When all phases are complete, we return the array of $k-1$ replacement path lengths. Our algorithm is summarized in Figure 2

The running time of our algorithm depends on the number of pivots executed during all UPDATE operations, the time required to find the next pivot, and the time required to answer DISTANCE queries. If we implement the UPDATE and DISTANCE operations using the parametric shortest-path algorithm of Young *et al.* [31], our generic

```
REPLACEMENTPATHS(G⁺):
  Compute the shortest path tree T_s in G⁺
  Preprocess G⁺ and T_s for UPDATE and DISTANCE
  for i ← k down to 1
      UPDATE(e_i, ∞)
      UPDATE(ℓ_i, W_i)
      R[i] ← DISTANCE(t)
  return R[1..k−1]
```

Figure 2. Our generic replacement paths algorithm.

algorithm runs in $O(mn + n^2 \log n)$ time, matching the time to run Dijkstra's algorithm in each graph $G \setminus e_i$.

In Section 4, we show that this generic algorithm can be implemented much more efficiently if the input graph is embedded. The next three lemmas, which actually hold for arbitrary graphs, prove useful in this analysis.

Lemma 3.1. *For any vertex v, the shortest path from s to v changes at most once during any* UPDATE *operation.*

Proof: Consider an UPDATE operation that *increases* the weight of an edge e. While the weight of e is increasing, the length of any shortest path that contains e is increasing, while the length of any shortest path that avoids e stays constant. Thus, if the shortest path from s to v changes, it must change from a path that contains e to a path that avoids e.

The other case is symmetric. While the weight of any edge e is decreasing, if the shortest path from s to v changes, it must change from a path that avoids e to a path that contains e. □

Lemma 3.1 immediately implies that at most $n-1$ pivots take place during each UPDATE operation. However, the special structure of our input graphs implies an even smaller number of pivots. In particular, the fact that t has no outgoing edges immediately implies the following:

Lemma 3.2. *At most one pivot occurs during each operation* UPDATE(ℓ_i, W_i) *in* REPLACEMENTPATHS.

Without assuming more about the graph, each operation UPDATE(e_i, ∞) could require $n-1$ pivots. However, even without additional assumptions, these pivots have a special structure that we will exploit in our later analysis. For any index i and any vertex $y \neq t$, let $Last(i, y)$ denote the index of the last vertex of P in the shortest path from s to y in G_i. Our assumption that all shortest paths are unique implies that the shortest path from s to y in G_i begins with the prefix $P[0..Last(i, y)]$ and otherwise avoids every edge of P.

Lemma 3.3. *For any vertex $y \neq t$ and any index $i > 1$, we have $Last(i-1, y) \leq Last(i, y)$.*

Proof: Let $dist_i(y)$ denote the shortest-path distance from s to y in G_i. Because $y \neq t$ and t has no outgoing edges, the shortest path from s to y avoids every edge ℓ_j. It follows that $dist_{i-1}(y) \geq dist_i(y)$, because G_{i-1} is a proper subgraph of G_i if all edges ℓ_j are ignored.

If $Last(i,y) < i-1$, the shortest path from s to y in G_i is a valid path in G_{i-1}, and therefore must be a *shortest* path in G_{i-1}, which implies that $Last(i-1,y) = Last(i,y)$. On the other hand, the definition of G_i implies that $1 \leq Last(i,y) < i$ for all i. Thus, if $Last(i,y) = i-1$, we immediately have $Last(i-1,y) < i-1 = Last(i,y)$. □

4 Surface-Embedded Graphs

Now we specialize our generic algorithm to solve the replacement path problem in surface-embedded graphs in $O(gn \log n)$ time. In particular, when the input graph is planar, our algorithm runs in $O(n \log n)$ time, matching Wulff-Nilsen's recent algorithm [30].

4.1 The Algorithm

Let G be a directed graph with non-negative edge weights, cellularly embedding on an orientable surface Σ of genus g. If the shortest path P happens to lie on the boundary of a single face of the embedding, then the augmented graph G^+ can also be embedded on Σ; in this case, we can solve the replacement path problem efficiently by combining our generic algorithm with Lemma 2.1 and bounding the number of pivots. In general, however, more work is required.

Recall from the previous section that each replacement path P_i is the concatenation of a prefix of P, a detour D_i, and a suffix of P. The first edge of D_i can leave P either to the left or to the right; similarly, the last edge of D_i can enter P either from the left or from the right. These alternatives give us four possible structures for the replacement path, illustrated in Figure 3. Following Emek *et al.* [6] and Wulff-Nilson [30], our algorithm separately computes replacement paths of each of these four types, and then combines the results.

To compute left-left replacement paths, we first construct a new graph LL from G by deleting every edge that either enters P from the right or leaves P to the right. (See Figure 4(a).) For each index i, the shortest left-left e_i-avoiding path from s to t in G is also the shortest e_i-avoiding path from s to t in LL. Because the entire path P lies on the boundary of a single face of LL, the augmented graph LL^+ is also embedded in Σ. Thus, we can compute left-left replacement paths efficiently by calling ReplacementPaths(LL^+), using the data structure described by Lemma 2.1 to implement the Update and Distance operations. We compute right-right replacement paths symmetrically.

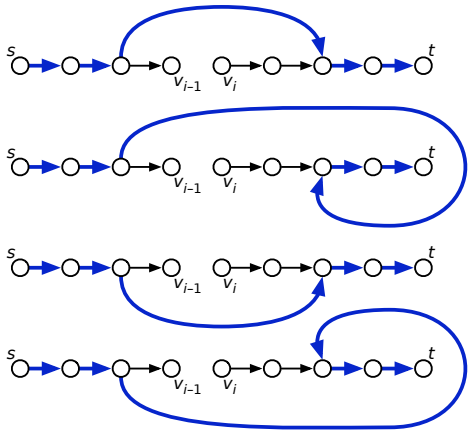

Figure 3. Four types of replacement paths in surface graphs. From top to bottom: left-left, left-right, right-right, and right-left.

The other two cases are slightly more complicated. To compute left-right replacement paths, we construct a graph LR by modifying G as follows. First, we delete all edges leaving P to the right or entering P from the left. Next, for each vertex v_i of P except s and t, we introduce a new vertex v_i', intuitively just to the right of P. Finally, we replace each edge $u \to v_i$ that enters P from the right with a new edge $u \to v_i'$. (See Figure 4(b).) Now a detour in G appears in LR as a path from some node v_i to some other node v_j'. Thus, before we augment LR to pass to our generic ReplacementPaths algorithm, we add edges $\ell_i = v_i' \to t$ for each index i. The resulting graph LR^+ still has an embedding in Σ, and ReplacementPaths(LR^+) correctly returns the lengths of the left-right replacement paths. Although the edges ℓ_i are defined differently than in Section 3, the proof of correctness and analysis of ReplacementPaths(LR^+) is completely unchanged. Again, we implement the Update and Distance operations using Lemma 2.1. We compute right-left replacement paths symmetrically.

4.2 Two Topological Lemmas

The analysis of our algorithm relies on the following pair of lemmas. Lemma 4.2 was previously used implicitly by several authors, including Klein [19], Cabello *et al.* [4, 5], Erickson [7], and Wulff-Nilsen [30]. Lemma 4.2 was used implicitly by Cabello *et al.* [4, 5]. For completeness, we include proofs of both lemmas here.

Lemma 4.1. *Let G be an undirected plane graph. Let T be an arbitrary spanning tree of G, let xy be an arbitrary edge of T, and let X be the component of $T \setminus xy$ containing vertex x. Finally, let f be an arbitrary face of G. Then X contains a (possibly empty) contiguous interval of vertices on the boundary of f.*

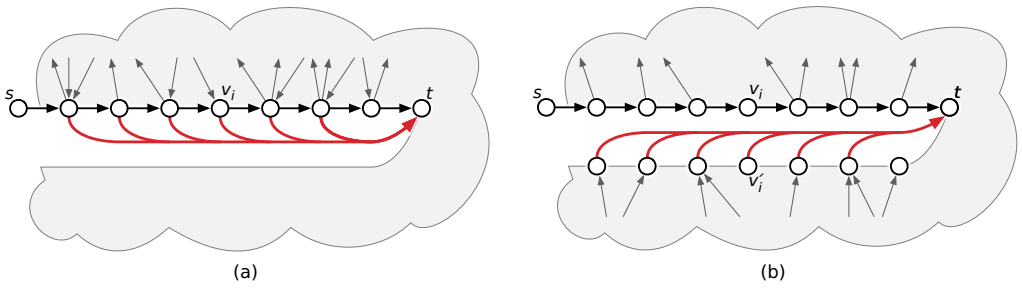

Figure 4. Graphs LL^+ and LR^+, with augmenting edges ℓ_j drawn concurrently.

Proof: Let $T[u,v]$ denote the unique path in T from vertex u to vertex v. Let a and c be vertices on the boundary of f, such that the paths $T[a,y]$ and $T[c,y]$ both end with the edge xy. (If two such vertices do not exist, the lemma is trivially true.) Vertices a and c partition the boundary of f into two paths, $\partial f[a,c]$ and $\partial f[c,a]$. Without loss of generality, suppose vertex y lies outside the cycle $\gamma := T[x,a] \cdot \partial f[a,c] \cdot T[c,x]$. For any vertex b on $\partial f[a,c]$, the path $T[b,x]$ cannot cross γ, and thus (by the Jordan Curve Theorem) cannot contain y. Thus, X contains every vertex of $\partial f[a,c]$. □

Lemma 4.2. *Let G be an undirected graph, embedded on an orientable surface Σ of genus $g > 0$. Let T be an arbitrary spanning tree of G, let xy be an arbitrary edge of T, and let X be the component of $T \setminus xy$ containing vertex x. Finally, let f be an arbitrary face of G. Then X contains at most $2g+1$ contiguous intervals of vertices on the boundary of f.*

Proof: Let Y be the component of $T \setminus xy$ containing y. Let $T[u,v]$ denote the unique path in T from vertex u to vertex v. Let G/f be the graph obtained from G by contracting the face f to a single vertex v_f. Let $T[u,v]/f$ denote the path in G/f corresponding to $T[u,v]$. Paths in X from x to f do not cross each other or paths in Y from y to f, so the corresponding paths in G/f also do not cross.

Let a and b be two vertices of X that lie on the boundary of f, such that the paths $T[x,a]/f$ and $T[x,b]/f$ are homotopic. Vertices a and c partition the boundary of f into two paths, $\partial f[a,b]$ and $\partial f[b,a]$. Without loss of generality, suppose the cycle $T[x,a] \cdot \partial f[a,b] \cdot T[b,x]$ is the boundary of a disk D. If this disk does not contain vertex y, then the Jordan Curve Theorem implies that X contains every vertex of $\partial f[a,b]$, exactly as in the planar setting.

On the other hand, if D does contain y, then D must also contain the entire subtree Y. Lemma now implies that Y contains a single (possibly empty) contiguous interval of vertices on the boundary of D, and therefore on the boundary of f. It follows that X also contains exactly *one* contiguous interval of vertices on the boundary of f in this case.

Now fix a subsequence $x_1, y_1, x_2, y_2, \ldots, x_k, y_k$ of $2k$ vertices on the boundary of f, such that each vertex x_i lies in X and each vertex y_i lies in Y. To compete the proof of the lemma, it suffices to show that $k \le 2g+1$. The two previous paragraphs imply that the paths $X[x,x_i]/f$ lie in distinct homotopy classes.

Let H denote the union of all paths $X[x,x_i]$; this is an acyclic subgraph of G, which inherits an embedding from G. The entire subtree Y—in particular, all vertices y_j—must lie in a common face of H. It follows that the induced embedding of H has exactly one face (which may or may not be a disk).

Finally, consider any *maximal* set of pairwise non-homotopic, non-crossing paths in Σ from x to v_f whose union does not disconnect Σ. Replacing each path with a single edge defines a new graph H' embedded on Σ, with exactly two vertices (x and v_f) and a single face. This face must be a disk; otherwise, we can add another edge, in a new homotopy class, that reduces its genus. Euler's formula $V - E + F = 2 - 2g$ implies that H' has exactly $2g+1$ edges.

We conclude that $k \le 2g+1$, which completes the proof of the lemma. □

We remark in passing that the bound $2g+1$ in the previous lemma is tight.

4.3 Analysis

We are finally ready to prove the main result of the paper.

Theorem 4.3. *Let G be a directed graph with non-negative edge weights, cellularly embedded on an orientable surface of genus $g > 0$. The replacement paths problem in G can be solved in $O(gn \log n)$ time.*

Proof: Lemma 2.1 implies that each of our four calls to ReplacementPaths runs in $O(n \log n + kg \log n + p \log n)$ time, where p is the total number of pivots occurring in all Update operations. Lemma 3.2 implies that

the operations UPDATE(ℓ_i, W_i) incur at most k pivots altogether. To complete the analysis, it remains only to bound the number of pivots occurring during the operations UPDATE(e_i, ∞). Lemma 3.1 implies that at most one edge incident to t pivots into T_s during any UPDATE operation, so we can safely ignore those edges.

When REPLACEMENTPATHS(G^+) is called, G^+ is one of the graphs LL^+, RR^+, LR^+, and RL^+. Let G^- be the subgraph of G^+ obtained by deleting all edges e_i and ℓ_i. Finally, for each index i, let T_i denote the shortest path tree in G^+ just after the operation UPDATE(e_i, ∞).

Fix an edge $x \to y$ of G^+ with $y \neq t$. Lemma 3.1 implies that $x \to y$ pivots into T_i at most once during any call to UPDATE. Thus, to count the number of times $x \to y$ pivots into T_s, it suffices to count indices i such that T_{i-1} contains $x \to y$ but T_i does not.

Let B_y denote the tree of shortest paths *ending* at y in G^-, and let B_x denote the subtree of B_y rooted at x. For any index i, the shortest path from s to y in G_i contains the shortest path from $v_{Last(i,y)}$ to y in G^-. Thus, $x \to y$ is an edge of T_i if and only if $v_{Last(i,y)}$ is a vertex of B_x. (Here we are exploiting our assumption that shortest paths are unique.) The vertices v_j of P all lie on a common face f of the embedding of G^-; thus, Lemma 4.2 implies that B_x contains at most $8g$ contiguous intervals of vertices in P. Lemma 3.3 now implies that there are at most $\max\{8g, 1\}$ indices i such that such that $v_{Last(i-1,y)} \in B_x$ but $v_{Last(i,y)} \notin B_x$. We conclude that $x \to y$ pivots into the shortest path tree T_s at most $8g$ times.

Altogether at most $8gn$ pivots occur during all operations UPDATE(e_i, ∞). Thus, our replacement paths algorithm runs in time $O(n \log n + kg \log n + gn \log n) = O(gn \log n)$, as claimed. □

For planar graphs, almost identical analysis implies a running time of $O(n \log n)$; the only difference is that we use Lemma 4.2 instead of Lemma 4.2.

Theorem 4.4 (Wulff-Nilsen [30]). *Let G be a directed planar graph with non-negative edge weights. The replacement paths problem in G can be solved in $O(n \log n)$ time.*

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. E. Wilber. Geometric applications of a matrix searching algorithm. *Algorithmica* 2:195–208, 1987.

[2] A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Tech. Report CS-TR-3539, University of Maryland Computer Science Department, 1995. ⟨http://www.lib.umd.edu/drum/handle/1903/763⟩.

[3] A. M. Bhosle. Improved algorithms for replacement paths problems in restricted graphs. *Oper. Res. Lett.* 33(5):459–466, 2005. ⟨http://www.cs.ucsb.edu/~bhosle/publications/repl-paths-orl.pdf⟩.

[4] S. Cabello and E. W. Chambers. Multiple source shortest paths in a genus g graph. *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 89–97, 2007.

[5] S. Cabello, E. W. Chambers, and J. Erickson. Multiple-source shortest paths in embedded graphs. Full version of [4], in preparation.

[6] Y. Emek, D. Peleg, and L. Roditty. A near-linear time algorithm for computing replacement paths in planar directed graphs. *Proc. 19th Ann. ACM-SIAM Symp. Discrete Algorithms*, 428–435, 2008.

[7] J. Erickson. Parametric shortest paths and maximum flows in planar graphs. *Proc. 21st Ann. ACM-SIAM Symp. Discrete Algorithms*, 794–804, 2010.

[8] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. *Discrete Comput. Geom.* 31:37–59, 2004.

[9] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *J. Algorithms* 34(2):251–281, 2000.

[10] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights. *Journal of Algorithms* 49(1):86–113, 2003.

[11] Z. Gotthilf and M. Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inform. Proc. Lett.* 109(7):352–355, 2009.

[12] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM* 46(4):502–516, 1999.

[13] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.* 55(1):3–23, 1997.

[14] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? *Proc. 42nd Ann. ACM Symp. Theory Comput.*, 252–259, 2001. Erratum in *Proc. 42nd Ann. ACM Symp. Theory Comput.*, 809, 2002.

[15] J. Hershberger, S. Suri, and A. Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms* 3(1):1–15, 2007.

[16] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM J. Comput.* 22(6):1199–1217, 1993.

[17] R. M. Karp and J. B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Appl. Math.* 3:37–45, 1981.

[18] P. Klein, S. Mozes, and O. Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *Proc. 20th Ann. ACM-SIAM Symp. Discrete Algorithms*, 236–245, 2009.

[19] P. N. Klein. Multiple-source shortest paths in planar graphs. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 146–155, 2005.

[20] K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Lett.* 8(4):223–227, 1989.

[21] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

[22] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica* 7(1):105–113, 1987.

[23] E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. *Inform. Proc. Lett.* 79(2):81–85, 2001.

[24] N. Nisan and A. Ronen. Algorithmic mechanism design (extended abstract). *Proc. 31st Ann. ACM Symp. Theory Comput.*, 129–140, 1999.

[25] L. Roditty and U. Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *Proc. 32nd Int. Colloq. Automata Lang. Program.*, 249–260, 2005. Lecture Notes Comput. Sci. 3580, Springer-Verlag.

[26] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.* 26(3):362–391, 1983.

[27] R. E. Tarjan. Dynamic trees via Euler tours, applied to the network simplex algorithm. *Mathematical Programming: Series A and B* 78:169–177, 1997.

[28] R. E. Tarjan and R. F. Werneck. Self-adjusting top trees. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 813–822, 2005.

[29] O. Weimann. *Accelerating Dynamic Programming*. Ph.D. thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 2009. ⟨http://www.wisdom.weizmann.ac.il/~oweimann/Publications/Thesis.pdf⟩.

[30] C. Wulff-Nilsen. Solving the replacement paths problem for planar directed graphs in $o(n \log n)$ time. *Proc. 21st Ann. ACM-SIAM Symp. Discrete Algorithms*, 756–765, 2010.

[31] N. E. Young, R. E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum balance algorithms. *Networks* 21(2):205–221, 1991.