Multiple-source shortest paths in planar graphs

Philip N. Klein*
Department of Computer Science
Brown University

1 Introduction

Given an n-node planar graph with nonnegative edge-lengths, our algorithm takes $O(n \log n)$ time to construct a data structure that supports queries of the following form in $O(\log n)$ time: given a destination node t on the boundary of the infinite face, and given a start node s anywhere, find the s-to-t distance.

The data structure requires $O(n \log n)$ space. To avoid using more than $O(n)$ space, if the pairs (s, t) are known in advance, the corresponding distances can be computed during the execution of the algorithm at a cost of $O(\log n)$ time per distance. The algorithm can also produce an $O(n)$ structure that, for any node s and any node t on the boundary, finds the first edge on the shortest s-to-t path in $O(\log \log \text{degree}(t))$ time. Using this structure, one can for example obtain the shortest s-to-t path P in time $O(|P|)$ if the graph has constant degree.

Using our algorithm, we obtain asymptotically faster algorithms for preprocessing to facilitate quick exact or approximate point-to-point distance queries in planar graphs (for arbitrary start and end nodes) and the corresponding shortest-path queries.

1.1 Previous work In the area of multiple-source shortest paths in planar graphs, there have been results of three kinds. We mention representative work. Frederickson [5] gave an $O(n^2)$ algorithm for all-pairs shortest paths. The special case where the boundary of an n-node planar graph has $O(\sqrt{n})$ nodes arises in applications of Lipton and Tarjan's planar separator theorem. [11] For this case, Fakcharoenphol and Rao [4] gave an $O(n \log^3 n)$ algorithm for computing what they called the *dense distance graph*, which is a recursive structure obtained using separators.[1] Finally, for the special case of a grid with arcs directed north and east and north-east, Schmidt [15] gave an $O(n \log n)$ algorithm that builds a data structure that, supports u-to-v distance queries in $O(\log n)$ time for any node u in in the west-most column of the grid and any node v.

1.2 Applications The multiple-source algorithm can be used to reduce the preprocessing time to construct data structures supporting exact and approximate point-to-point distance queries (i.e., for any nodes u and v, "what is the u-to-v distance?") Fakcharoenphol and Rao show that their dense distance graph supports $O(\sqrt{n} \log^2 n)$ queries. We improve the construction time for this graph from $O(n \log^3 n)$ to $O(n \log^2 n)$ for the special case of nonnegative lengths. Fakcharoenphol and Rao give a dynamic algorithm that supports point-to-point distance queries and single-edge length-changes in amortized $O(n^{2/3} \log^{7/3} n)$ time per operation. Our algorithm yields a simpler dynamic algorithm for which the worst-case time per operation is $O(n^{2/3} \log^{5/3} n)$. See Section 6.

Thorup [17] shows how to construct data structures supporting fast ϵ-approximate point-to-point distance queries. He gives a preprocessing algorithm requiring $O(n\epsilon^{-1} \log^2 n \log \Delta)$ time to build a structure supporting queries in $O(\log \log \Delta + \epsilon^{-1} \log n)$ time, where Δ is an upper bound on the largest finite distance. We show how to improve the preprocessing time to $O(n(\epsilon^{-1} + \log n) \log n \log \Delta)$. See Section 7. (Thorup also gives a structure supporting faster queries, but the preprocessing time is higher, and our method seems inapplicable.)

Once an exact or approximate distance query has been answered, our algorithm can be used to find

*Partial support provided by NSF Grant CCR-97000146

[1] Note, however, that their algorithm solved a more difficult problem, one where negative lengths are allowed.

the corresponding path P in time $O(|P|)$ for constant-degree graphs.[2] No additional preprocessing time or storage are required.[3]

1.3 Related work Ripphausen-Lipa, Wagner, and Weihe [14, 19] have given algorithms for flow problems in planar graphs using an approach that considers orientation and crossing and rightmost paths, ideas that inform the present work.

Eppstein, Italiano, Tamassia, Tarjan, Westbrook, and Yung [3] give a dynamic algorithm to maintain a minimum spanning forest in a planar embedded graph in the presence of edge deletions and insertions. They use a dynamic tree in the primal and the dual graphs, an idea we use in the present paper.

2 Background and terminology

In this paper we are concerned with *directed* graphs. If y is a node on a path P that starts at x, the *to-y prefix* of P is the x-to-y subpath of P. The *from-y suffix* of P is defined similarly. A path is *simple* if no node occurs more than once in the path. A nonsimple path contains a cycle. One can *remove* cycles from a nonsimple u-to-v path P, obtaining a simple u-to-v path whose edge-set is a proper subset of that of P. Given an r-rooted tree T and a node v, $T[v]$ denotes the simple path from r to v in T (which is unique if it exists). If u is an ancestor of v (or if T is considered as an undirected tree), $T[u, v]$ denotes the simple u-to-v path in T.

2.1 Shortest paths In this section, we discuss background concerning shortest paths in arbitrary directed graphs. We use $\ell(e)$ to denote the length of an edge e. For a path or cycle P, $\ell(P) = \sum_{e \in P} \ell(e)$. We assume no cycle has negative length.

It follows from this assumption that removing cycles from a nonsimple path does not increase its length. Hence if there is a shortest u-to-v path, there is one that is simple.

For any labeling $d(\cdot)$ of the nodes of a graph with numbers, there is a new length assignment $\hat{\ell}$ defined by $\hat{\ell}(uv) = d(u) + \ell(uv) - d(v)$

We refer to the lengths $\hat{\ell}(uv)$ as *reduced lengths with respect to d*. For any nodes s and t, for any s-to-t path P,

$$\hat{\ell}(P) = \ell(P) + d(s) - d(t) \tag{2.1}$$

It follows that an s-to-t path is shortest according to $\hat{\ell}$ if and only if it is shortest according to ℓ.

An edge uv is a *relaxed edge* with respect to d if its reduced length is nonnegative, i.e. if

$$d(u) + \ell(uv) \leq d(v) \tag{2.2}$$

and is an *unrelaxed edge* otherwise. An edge is *tight* with respect to d if its reduced length is zero, i.e. if (2.2) is satisfied with equality.

Suppose that every edge is relaxed and, for some nodes r and v, $d(r) = 0$ and $d(v)$ is the length of *some* r-to-v path P_v. It follows by (2.1) that $\hat{\ell}(P_v) = 0$. Every edge and hence every r-to-v path has nonnegative reduced length, so P_v is a shortest path with respect to the reduced lengths $\hat{\ell}$, and hence also with respect to the original lengths ℓ.

2.1.1 Network simplex Our algorithm, like the network-simplex algorithm,[4] maintains an r-rooted directed spanning tree T. For each node $v \neq r$ reachable from r, there is a unique incoming edge in T, called the *parent edge* of v in T. The tree induces an assignment $d_T(\cdot)$ of distance estimates, namely $d_T(v) = \ell(T[v])$. For each edge uv in T, $\ell(T[v]) = \ell(T[u]) + \ell(uv)$, whence $d_T(v) = d_T(u) + \ell(uv)$, so uv is tight.

The nonexistence of negative-length cycles implies that v is not an ancestor of u in T if uv is unrelaxed. A pivot step consists of selecting an unrelaxed edge uv and modifying T by removing the parent edge of v and adding the edge uv. We call this step *relaxing* the edge uv. If there are no edges that are unrelaxed with respect to T, it follows that the distance estimates $d_T(v)$ are true distances from r. In this case T is a *shortest-path tree.*

2.2 Planar embeddings We assume basic knowledge of planar embedded graphs, faces, and the dual. For a node v on the boundary of the infinite face z, it would not violate planarity to embed an artificial node $\hat{v}$ and an artificial edge $\hat{v}v$, both inside z. For

[2] More generally, the time is $O(\log\log \text{degree}(x))$ per node x in the path.

[3] Thorup showed how to achieve $O(|P|)$ time for approximate shortest paths, but his approach requires an additional $O(\log n)$ factor in time and space.

[4] We are indebted to F. Barahona for pointing out the simplex interpretation of our algorithm.

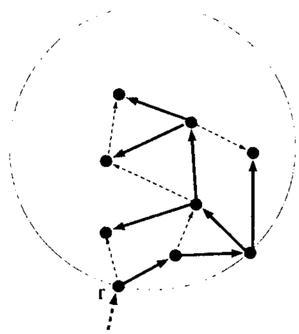

Figure 1: The boundary of the infinite face is indicated by the circle. A right-first search tree rooted at r is indicated by the bold edges. For each node v other than the root, the parent edge is the edge by which v was first visited.

a node v with incoming edge uv and outgoing edges vw and vx, we say vx is *left of vw with respect to uv* if vx occurs strictly between vw and uv in counter-clockwise order. Given a simple path P containing an edge vw, we say an edge vx *emanates* left from P if (a) the edge preceding vw in P is uv, and vx is left of vw with respect to uv, or (b) v is the first node of P, and v lies on the boundary of the infinite face, and vx is left of vw with respect to the artificial edge $\hat{v}v$.

For a primal edge e, the corresponding dual edge points from e's left face (the face to e's left when e is oriented upwards) to e's right face.

3 Ingredients and the algorithm

3.1 Rightmost shortest-path tree *Right-first search* [14] is depth-first search on a planar graph, with the restriction that, for each node v visited, the edges vw out of v are explored in right-to-left order with respect to the the edge uv by which v was first visited (or, if v is the root and is on the boundary of the infinite face, with respect to the artificial edge $\hat{v}v$). Right-first search induces a *right-first search tree* consisting of the set of such edges uv (see Figure 1).

The *rightmost* shortest-path tree rooted at r_0 (formally defined in Section 4.3) can be obtained from the set of from-r_0 distances $d(\cdot)$ by finding a right-first search tree in the subgraph of edges that are tight with respect to $d(\cdot)$.

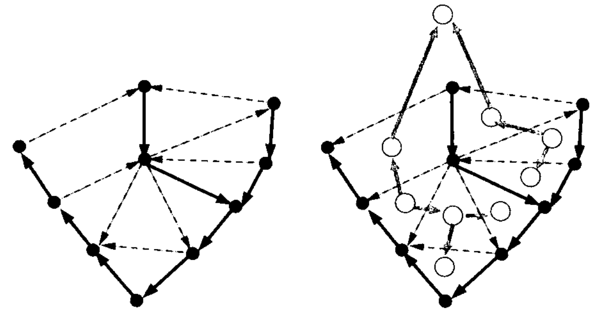

Figure 2: On the left is a primal graph. A spanning tree is indicated in bold. On the right is shown the primal graph and the dual graph not including the nontree dual edges.

3.2 Leafmost unrelaxed edge Let T^* denote the set of edges *not* in the current tree T. As noted in Subsection 2.1, every edge of T is relaxed, so T^* includes all unrelaxed edges. The fact that T is a spanning tree of the planar primal implies that T^* is a spanning tree of the planar dual (ignoring edge orientations). Consider T^* as a dual spanning tree rooted at the node of the planar dual corresponding to the infinite face (see Figure 2). A *leafmost unrelaxed edge xy* is an unrelaxed edge none of whose proper descendent edges in T^* is unrelaxed.

3.3 The algorithm Let $r_0, r_1, \ldots, r_s$ be the nodes on the boundary of the infinite face, in clockwise order. First we add auxiliary edges[5] of infinite length: $r_s r_{s-1}, r_{s-1} r_{s-2}, \ldots, r_2 r_1, r_1 r_0$. We then carry out the following:

let T be a *rightmost* shortest-path tree rooted at r_0.
for $i := 1, \ldots, s$,
    remove the edge of T entering r_{i-1},
    and add $r_i r_{i-1}$.
    (Now T is rooted at r_i.)
    While there exists an unrelaxed edge,
        relax a leafmost unrelaxed edge.

Our pivot selection rule is to choose a *leafmost unrelaxed edge*. The motivation for this selection rule is as follows. Let e be an edge not in the primal spanning tree T. There is a unique simple undirected

[5]These can be added without destroying planarity since the r_i's are on the boundary of the infinite face.

path in T connecting e's endpoints, which together with e forms a simple cycle C_e in the primal. The nontree edges embedded interior to C_e are precisely the strict descendents of e in the dual spanning tree rooted at the infinite face. In particular, if e is a leafmost unrelaxed edge then no unrelaxed edges are strictly interior to C_e.

We will show that the algorithm takes $O(n \log n)$ time. In Section 4, we show that each edge is relaxed at most once. In Section 5, we show how each iteration can be implemented in $O(\log n)$ time where n is the number of nodes. At the end of iteration i of the for-loop, the current tree is an r_i-rooted shortest-path tree. We note in Section 5 that the distance in this tree from r_i to any node can be queried in $O(\log n)$ time. In addition, by using the persistence technique of [2], the algorithm's history can be recorded in $O(n \log n)$ space so as to permit the subsequent querying of any of the shortest-path trees.[6]

Each edge appears in the shortest-path trees of a contiguous subsequence of the cycle of roots around the boundary of the planar graph. This fact follows from the fact that each edge is relaxed at most once, and can be proved more directly without reference to the algorithm. It generalizes a lemma of Frederickson (see [6]) for outerplanar graphs. It implies a size-$O(n)$ representation of multiple shortest-path trees: for each node v, organize [12, 18] the outgoing edges according to the disjoint intervals of roots whose shortest-path trees they belong to. Given a node v and a boundary node r_i, one can find the first edge on the shortest v-to-r_i path.

4 Analysis

4.1 Flows, potentials, and circulations The material in this subsection is adapted from [10] except for the lemma, which is new.

For a graph G, an *integral flow assignment* f is a function from the edges of G to the integers. We adopt the *antisymmetry convention*: if xy is an edge, $f(yx)$ is defined to be $-f(xy)$. Thus f assigns integral flow values to the edges and the reverses of edges. For an edge e, let e^R denote the reverse of e. For a path P, let P^R denote the reverse path, i.e. consisting of the reverses of the edges of P in the reverse order.

[6] I am indebted to R. Tarjan for this observation.

Note that flow assignments can be added and subtracted. For flow assignments f_1 and f_2, the flow assignment $f_1 + f_2$ assigns $f_1(xy) + f_2(xy)$ to the edge xy.

For G an embedded planar graph, let ϕ be a function from the faces of G to the integers such that the infinite face maps to zero.[7] We call $\phi(z)$ the *potential* of face z. We call ϕ a *potential function*. The corresponding flow assignment is

$$f_\phi(e) = \phi(\text{face to } e\text{'s right}) - \phi(\text{face to } e\text{'s left})$$

This flow assignment is a *circulation*, i.e. for each node v, $\sum_u f(uv) = 0$ where the sum is over all nodes u for which $f(uv)$ is defined (i.e. uv or $(uv)^R$ is an edge).

The sum of circulations corresponds to the sum of potential functions, i.e. for potential functions ϕ_1 and ϕ_2, the sum $f_1 + f_2$ of the corresponding flow assignments corresponds to the sum $\phi_1 + \phi_2$ of potential functions.

We say the circulation is *clockwise* if every potential is nonnegative, and *counterclockwise* if every potential is negative. For example, for a clockwise simple cycle C of edges and reverses of edges, the circulation assigning 1 to the edges/reverse edges of C and zero to all others corresponds to assigning a potential of 1 to every face enclosed by C and 0 to all other faces, and hence the circulation is clockwise.

LEMMA 4.1. *Consider a potential assignment ϕ. If the corresponding flow assignment f contains a counterclockwise simple cycle of positive flow that is not enclosed in a clockwise simple cycle of positive flow, the circulation is not clockwise.*

4.2 "Is left of" and "is right of" Weihe [19] defined a relation "is more left than" between s-t flows in a planar directed graph. We specialize his definition to get a relation between s-to-t paths. A path P corresponds to a flow assignment f_P that assigns 1 to each edge of P and zero to other edges. For s-to-t paths P and Q, we say P is *left of* Q (and Q is *right of* P) if $f_P - f_{Q^R}$ is a clockwise circulation. It is straightforward to show that the "is left of" relation is reflexive, transitive, and antisymmetric.

[7] In [10], there is a distinguished face. In this paper, we use the infinite face as the distinguished face, and change the terminology accordingly.

LEMMA 4.2. (DISJOINTNESS) *If P_1 and P_2 are simple u-to-v paths that share no nodes except u and v, then either P_1 is to the left of P_2 or vice versa.*

LEMMA 4.3. (CONCATENATION) *Let P_1 and P_2 be simple u-to-x paths, and let P'_1 and P'_2 be simple x-to-v paths. If P_2 is left of P_1 and P'_2 is left of P'_1 then $P_2P'_2$ is left of $P_1P'_1$.*

LEMMA 4.4. *Suppose P and Q start and end at the same nodes, and their common start node is on the boundary of the infinite face. Then P is left of Q iff no edge of Q emanates left from P.*

We define "left of" for spanning trees in terms of "left of" for paths. For two r-rooted spanning trees T_1 and T_2, we say T_1 is *left of* T_2 if, for every node v, the path $T_1[v]$ is left of $T_2[v]$.

4.3 Rightmost shortest-path tree An r-rooted shortest-path tree is a *rightmost* shortest-path tree if in addition every other r-rooted shortest-path tree is left of T. A r-rooted rightmost search tree T of a graph G has the property [14] that, for every node v, $T[v]$ is a rightmost r-to-v path in G. It follows that the r-rooted rightmost search tree of the subgraph of G consisting of tight edges is a rightmost shortest-path tree.

4.4 The *right-shortness* invariant, and maintaining it while relaxing edges We define an r-rooted tree T to be *right-short* if the following condition holds for every node v: if P is a simple r-to-v path that is right of $T[v]$ and $\ell(P) \leq \ell(T(v))$ then $P = T[v]$. That is, there is no simple r-to-v path strictly right of $T[v]$ that is as short as $T[v]$ itself.

A nontree edge xy is *left-to-right* (with respect to T) if the r-to-y path consisting of $T[x]$ and xy is left of the r-to-y path $T[y]$.

LEMMA 4.5. *Suppose T is an r-rooted right-short tree, and e is an unrelaxed edge. Then e is left-to-right with respect to T.*

THEOREM 4.1. (BASIC STEP) *Suppose T is an r-rooted right-short tree, and e is a leafmost unrelaxed edge. Then relaxing e yields an r-rooted right-short tree T' that is left of T.*

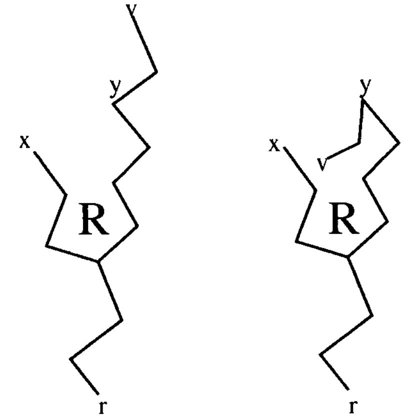

Figure 3: Two possible embeddings. The bold edges are edges of T. The light edge is the edge xy relaxed to obtain T'. The last node common to $T[x]$ and $T[v]$ is z. The node v is a descendent of y in T. R is the region bounded by the z-to-x and z-to-y paths in T and the edge xy.

Proof. Let $e = xy$. We first prove that T' is left of T. By Lemma 4.5, $T'[y]$ is left of $T[y]$. Let v be any node. If v is not a descendent of y in T then $T'[v] = T[v]$, so $T'[v]$ is left of $T[v]$. Suppose v is a descendent of y, and let P be the y-to-v path in T (which is also the y-to-v path in T'). Then $T[v] = T[y]P$ and $T'[v] = T'[y]P$. It follows from Lemma 4.3 that $T'[v]$ is left of $T[v]$.

Now we prove that T' is right-short. Assume for a contradiction that, for some node v, there is a simple r-to-v path P that is right of $T'[v]$ and distinct from $T'[v]$ such that P has length no more than $T'[v]$. Choose P to be the shortest such path. If $T'[v] = T[v]$ then P would violate the right-shortness of T. Hence $T'[v]$ must use the one edge in $T' - T$, namely xy, the edge relaxed. It follows that v must be a descendent of y in T', and therefore in T as well. (See Figure 3.)

First suppose that $P = P_1xyP_2$ for some paths P_1 and P_2. By Lemma 4.4, P contains no edge that emanates from the left of $T'[v]$, so no edge of P_1 emanates left of $T'[x]$ and no edge of xyP_2 emanates left of $xyT[y, v]$. The former implies, again by Lemma 4.4, that $T'[x]$ is left of P_1. But $T'[x] = T[x]$, so $T[x]$ is left of P_1. By right-shortness of T, it follows that either $P_1 = T[x]$ or $\ell(P_1) > \ell(T[x])$.

Let $P'_2 = T[y]P_2$. Since no edge of xyP_2 emanates left of $xyT[y, v]$, and $T[v] = T[y]T[y, v]$, no edge of P'_2 emanates left of $T[v]$, so $T[v]$ is left of P'_2. Hence either $P'_2 = T[v]$ or $\ell(P'_2) > \ell(T[v])$. Since $\ell(P'_2) = \ell(T[y]) + \ell(P_2)$, either $P_2 = T[y, v]$ or $\ell(P_2) > \ell([y, v])$.

Combining these two facts, we infer that either $P = T[x]xyT[y, v]$ or $\ell(P) > \ell(T[x]) + \ell(xy) + \ell(T[y, v])$. Recall that $T'[v] = T[x]xyT[y, v]$. We assumed $P \neq T'[v]$ and $\ell(P) \leq \ell(T'[v])$, so this is a contradiction.

Hence P cannot contain the edge xy. The following claim shows that $\ell(P) \geq \ell(T[v])$. Since $\ell(T[v]) > \ell(T'[v])$, this contradicts the choice of P, completing the proof of the theorem.

CLAIM 4.5.1. *For each node u of P that also appears on $T[v]$, the to-u prefix of P is no shorter than $T[u]$.*

The proof of the claim is by induction. The claim is trivial for the root r. Let $u \neq r$ be a node of P on $T[v]$, and let w be the previous node of P that is also on $T[v]$. By the inductive hypothesis, the to-w prefix of P is no shorter than $T[w]$.

Let P_1 be the w-to-u subpath of P. Using the inductive hypothesis, the length of the to-u prefix of P is no less than $\ell(T[w]) + \ell(P_1)$. Note that u and w are the only nodes common to P_1 and $T[v]$. By the Disjointness Lemma, there are three cases, corresponding to the relative placement of P_1 and $T[v]$. Case 1 holds when u occurs before w in $T[v]$, Case 2 holds when P_1 is to the right of $T[w, u]$, and Case 3 holds when P_1 is to the left of $T[w, u]$ and does not coincide with $T[w, u]$. Cases 2 and 3 are illustrated in Figure 4.

Case 1: If $\ell(T[w]) + \ell(P_1)$ were less than $\ell(T[u])$, then P_1 together with $T[u, w]$ would form a negative length cycle, a contradiction.

Case 2: In this case P_1 is right of $T[w, u]$. Let $P_2 = T[w]P_1$. By the Concatenation Lemma, P_2 is right of $T[u]$. The right-shortness of T implies that $\ell(P_2) \geq \ell(T[u])$. It follows that $\ell(P_1) \geq \ell(T[w, u])$. Combining this with the inductive hypothesis completes the inductive step.

Case 3: In this case P_1 is left of $T[w, u]$. We first show that every edge of P_1 lies in the region R enclosed by the edge xy and the simple undirected path in T connecting x and y. Suppose not, and let st be the first edge of P_1 not in R. The boundary of R consists of a subpath of $T'[v]$ on the left and a subpath of $T[v]$ on the right. Since P_1 is internally node-disjoint from $T[v]$, the node s must belong to $T'[v]$ and st must emanate to the left of $T'[v]$. This contradicts the fact that P contains no such edge.

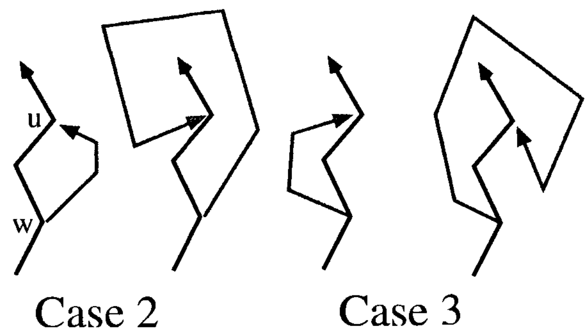

Figure 4: This figure illustrates examples of the cases in the induction proof. In each case, the bold path is $T[v]$ and the light path is P_1. The two leftmost drawings represent Case 1, in which P_1 is to the right of the subpath of $T[v]$, and the two rightmost drawings represent Case 2, in which P_1 is to the left.

We conclude that every edge of P_1 lies in the region R. By the leafmost selection rule, every edge in this region except xy itself is relaxed, and P_1 does not contain xy. Hence every edge of P_1 is relaxed. Let the nodes of P_1 be $w = z_0, z_1, \ldots, z_k = u$, and let α_i denote the length of the to-z_i prefix of P. The inductive hypothesis states that $\alpha_0 \geq d_T(z_0)$. Assuming $\alpha_{i-1} \geq d_T(z_{i-1})$, since $d_T(z_i) \leq d_T(z_{i-1}) + \ell(z_{i-1}z_i)$, we get $\alpha_i \geq d_T(z_i)$. For $i = k$, then, the length of the to-u prefix of P is $\geq d_T(u)$. ■

4.5 Applying right-shortness We show by induction that every tree T arising in the algorithm is right-short. The initial tree is a rightmost shortest-path tree, so is trivially right-short. Suppose that at the beginning of iteration i the tree T rooted at r_{i-1} is right-short. Lemma 4.6 shows that the modification to obtain an r_i-rooted tree preserves right-shortness. The Basic-Step Theorem shows that each iteration of the inner loop preserves right-shortness.

LEMMA 4.6. *Suppose T is a right-short tree rooted at r_{i-1}, and T' is obtained by removing the parent edge of r_i and adding the edge r_ir_{i-1}. Then T' is right-short.*

Proof. Suppose P is an r_i-to-v path to the right of $T'[v]$, and P is no longer than $T'[v]$. We claim that

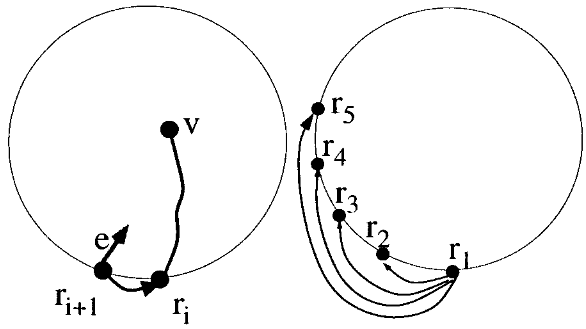

Figure 5: In both figures, the circle represents the boundary of the infinite face. In the left figure, the bold path is $T_{i+1}[v]$, which starts with the edge $r_{i+1}r_i$, and the light edge is e. In the right figure, the solid arrows denote the auxiliary edges

the first edge e of P is $r_i r_{i-1}$. Once we prove the claim, note that the rest of P is to the right of $T[v]$ and is no longer than $T[v]$, hence is itself $T[v]$. The Concatenation Lemma implies that P is itself $T'[v]$.

To prove the claim, assume for a contradiction that $e \neq r_i r_{i-1}$. Then e must be embedded as shown on the left of Figure 5. Lemma 4.4 shows that P is not to the right of $T^0_{i+1}[v]$. ■

LEMMA 4.7. *Let $\hat{T}$ be the initial shortest-path tree, let T be a any tree arising in the algorithm, and let v be any node. No edge of $T[v]$ emanates left of $\hat{T}[v]$.*

Proof. Suppose an edge xy of $T[v]$ emanates left of $\hat{T}[v]$, and let z be the first node of $\hat{T}[v]$ to occur after x on $T[v]$. Since $\hat{T}[x,z]$ and $T[x,z]$ are internally node-disjoint, by the Disjointness Lemma either $\hat{T}[x,z]$ is left of $T[x,z]$ or vice versa. By Lemma 4.4, $T[x,z]$ is left of $\hat{T}[x,z]$. Let $P = T[x]\hat{T}[x,z]$. By the Concatenation Lemma, P is right of $T[z]$. However, because $\hat{T}$ is a shortest-path tree, $\hat{T}[x,z]$ is a shortest x-to-z path, so $\ell(P) = \ell(T[x]) + \ell(\hat{T}[x,z]) \leq \ell(T[x]) + \ell(T[x,z]) = \ell(T[z])$, contradicting the right-shortness of T. ■

For the purpose of analyzing the algorithm, embed an artificial node z and artificial edges $zr_0, zr_1, zr_2, \ldots, zr_s$ in the infinite face, preserving planarity. Suppose the algorithm performs k relaxations in total. For $0 \leq i \leq k$, let T_i denote the tree T after i relaxations, modified by adding the artificial edge from z to the root of T. For any node v and $0 \leq i \leq k$, let f_i^v denote the flow corresponding to $T_i[v]$. Let ϕ_{ij}^v denote be the potential function corresponding to the circulation $f_i^v - f_j^v$. We leave out the superscript when it is clear.

COROLLARY 4.1. *If $i \geq j$, ϕ_{ij}^v assigns at most 1 to every face.*

Proof sketch. A contradiction would give rise to two cycles C_1 and C_2 of flow, the former enclosing the latter. One then shows there is an edge e_1 of $(T_j[v])^R$ on or enclosed by C_2 whose successors in $(T_j[v])^R$ are all external to C_2. Hence the successor of e_1 in C_2 emanates left from $T_j[v])^R$, contradicting Lemma 4.7. ■

THEOREM 4.2. *For any edge e, the set $\{i : e \in T_i\}$ is a consecutive subsequence of the cycle* $(0\ 1\ \ldots\ k)$.

Proof. Assume the theorem is false for $e = uv$, so there exist integers $0 \leq a < b < c < d \leq k$ such that either $e \in T_a, T_c$ and $e \notin T_b, T_d$ or $e \in T_b, T_d$ and $e \notin T_a, T_c$. Assume the former without loss of generality. Let e' be the last edge in $T_b[v]$. Then the circulation $f_b^v - f_a^v$ contains one edge entering v, namely e', and one edge leaving v, namely e. The circulation $f_c - f_b$ contains one edge entering v, namely e, and one edge leaving v, namely e'. Hence the circulation $f_c - f_b + f_b - f_a$ contains no edges incident to v.

Note that ϕ_{ba}^v and ϕ_{cb}^v each assign potential 1 to some faces that have v on their boundary. Since the circulation $f_c - f_b + f_b - f_a$ contains no edges incident to v, the corresponding potential ϕ_{ca} assigns potential 1 to all the faces that have v on their boundary.

The circulation $f_d - f_a$ contains one edge entering v and one edge (namely e) leaving v. However, this circulation corresponds to the potential $\phi_{dc} + \phi_{ca}$. Since ϕ_{ca} assigns 1 to all faces that have v on their boundary, and ϕ_{dc} assigns only nonnegative potentials (since P_d is left of P_c) and by Corollary 4.1 the potential $\phi_{da} = \phi_{dc} + \phi_{ca}$ assigns at most 1 to each face, it follows that ϕ_{da} corresponds to a circulation that contains no edges incident to v. This contradiction completes the proof. ■

It follows that each edge gets relaxed at most once, for a total of $O(n)$ relaxations. It remains to show that each iteration can be implemented in $O(\log n)$ time.

5 Implementing a basic step

Our data structure consists of two representations of spanning trees, one for the modified input graph (the primal), representing T, and one for its planar dual, representing T^*, the spanning tree of the dual graph consisting of the edges not in T. The primal spanning tree is used for computing shortest-path distances from the various roots, and the dual spanning tree is used to locate edges that need to be relaxed.

A dynamic-tree data structure [16] represents a set of rooted or unrooted trees under structure-modifying and weight-related operations at $O(\log n)$ time per operation. Top trees [1] build on [16] via topology trees [8] and make new operations easy to implement.

The structure-modifying operations are: link(v, w) where v and w are nodes of different trees, links the trees by adding the edge vw, and cut(e), which removes the edge e. One can also obtain for any node v the parent edge of v.

The data structure can also maintain weights on nodes/edges. For the primal tree, we use the edge-lengths as weights. The operation we need is sum(x), which returns the sum of weights on the root-to-x path in the forest. This is used to find the distance from r_i to any desired node as mentioned in Section 3.3.

For the dual, we maintain an implicit representation of the reduced lengths of the edges in T^* (the edges not in T^* have reduced length zero). Note that edges in T^* are not oriented consistently, so paths to the root can have edges in both directions. One operation needed is find(), which returns a leafmost unrelaxed edge e in T^* (i.e. $\hat{\ell}(e) < 0$ and, for each proper descendent edge e' of e, $\hat{\ell}(e') \geq 0$) and its reduced length $\hat{\ell}(e)$. The other operation is change(x, Δ), which changes the $\hat{\ell}$ values of all edges e on the path between x and the root as follows:

$$\hat{\ell}(e) := \hat{\ell}(e) + \begin{cases} \Delta & \text{if } e \text{ points towards root} \\ -\Delta & \text{if } e \text{ points away from root} \end{cases}$$

To implement a basic step, the algorithm proceeds as follows. Use find to find a leafmost unrelaxed edge uv. Let $\Delta = -\hat{\ell}(uv)$. Let wv be the parent edge of v in T, and let $xy = (wv)^*$ be the corresponding dual edge.

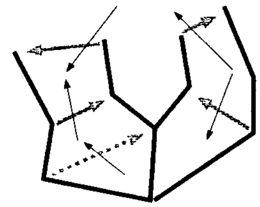

Figure 6: The dark edges are in T. The dashed edge is being relaxed, changing the distances to nodes of the inner tree. This requires changing the reduced lengths of edges to and from the inner tree. Edges pointing to the inner tree from the left correspond to dual edges pointing toward the root.

The algorithm must update the $\hat{\ell}$ values to reflect a reduction by Δ in the distance estimates of nodes in the primal tree's v-rooted subtree T'. Values for edges pointing from nodes not in T' to nodes in T' should increase by Δ, and edges pointing in the opposite direction should decrease by Δ. See Figure 6. Let z denote the least common ancestor of x and y. By the right-hand-rule, a left-to-right nontree edge of the primal corresponds to a dual edge that points towards the root in the dual spanning tree. Therefore the dual edges whose values must increase are those edges in $T^*[x, z]$ that point towards the root and those in $T^*[y, z]$ that point away. The edges whose values must decrease are the edges in these same paths but pointing the opposite direction. The algorithm therefore calls changeValue(x, Δ) and changeValue$(y, -\Delta)$. This achieves the desired changes, leaving unchanged the values on the edges in the undirected path between z and the root.

Next the algorithm uses cut and link operations to change the primal and dual trees to reflect the substitution of primal edge uv for wv.

6 Exact point-to-point distances

We are given a planar embedded graph G and c Jordan curves whose strict interiors are disjoint and are contained in faces of G. The Jordan curves intersect no edges, and intersect $O(\sqrt{n})$ nodes, called *border nodes*. The task is to compute all border-node-to-border-node distances. For each Jordan curve J, use the multiple-source algorithm to find distances from/to border nodes on J to/from all other border nodes in $O(c(n + \sqrt{n}^2) \log n)$ time.

By repeated use of this algorithm, one can compute the *dense distance graph* defined by Fakcharoenphol and Rao [4] in $O(n \log^2 n)$ time. They show that this structure supports exact point-to-point distance queries.

For a dynamic algorithm, divide [5, 13] the graph into $O(n/r)$ edge-disjoint regions each with $O(r)$ nodes and $O(\sqrt{r})$ border nodes (nodes belonging to other regions) such that border nodes in a region lie on $O(1)$ faces. The dynamic algorithm maintains for each region (1) an implicit representation of all x-to-y distances where either x or y is a border node, and (2) explicit distances where x and y are both border nodes. When an edge's length changes, the algorithm recomputes (1) and (2) for the region containing that edge in $O(r \log r)$ time. To compute u-to-v distance: (A) Assuming u is not a border node, compute the distances within u's region from u to the set S_u of border nodes of this region. (B) Run Fakcharoenphol and Rao's implementation of Dijkstra's algorithm, initialized with the distances computed for S_u, obtaining distances $d(\cdot)$ in G to all border nodes. (C) Assuming v is not a border node, compute the distances in v's region from the border nodes of that region to v, and combine this with the distances $d(\cdot)$ assigned to these nodes to obtain the u-to-v distance.

The time for the query algorithm is $O((n/\sqrt{r}) \log^2 n)$, and the time for the update algorithm is $O(r \log r)$. We can choose r to get $O(n^{2/3} \log^{2/3})$ time for queries and updates, improving on the previous bound by a factor of $\Theta(\log^{5/3})$.

7 Approximate point-to-point distances

Let G be an n-node planar graph G with a shortest path $P = r_s \dots r_0$ of length $\leq \alpha$ along a face boundary, where α and ϵ are parameters. For $i = 0, \dots, s$, define $\gamma(r_i)$ = length of subpath $r_s \dots r_i$ of P, and define $\delta_i(v) = \gamma(i) + r_i$-to-$v$ distance. A set S of pairs (r_i, v) called *connections* is said to *cover* a vertex v if S contains some connection (r_{i^*}, v) such that $\delta_{i^*}(v) \leq \min_i \delta_i(v) + \epsilon\alpha$. The core problem is to find a set S of connections that cover all nodes v such that $\min_i \delta_i(v) \leq 2\alpha$. We give an algorithm to find such a set S such that $|S| = O(n(\epsilon^{-1} + \log n))$. Using Lemma 18 of [17], S can then be pruned in $O(|S|)$ time to contain $O(\epsilon^{-1})$ connections per node.

To compute S, we run an augmented version of the multiple-source algorithm. The algorithm starts with the shortest-path tree rooted at r_0, and in successive iterations of the for-loop computes the shortest-path trees rooted at $r_1, \dots, r_s$. We augment the algorithm so that, in each such iteration i, it identifies some nodes v and adds the connections (r_i, v) to S.

To this end, the algorithm maintains an implicit representation of a node-labeling $\sigma(\cdot)$giving the amount by which the δ distance of v must decrease for there to be a new connection involving v.

To initialize, for each node v, if $\delta_0(v) \leq 2\alpha$, a connection (r_0, v) is added to S and $\sigma(v)$ is assigned $\epsilon\alpha$; otherwise, $\sigma(v)$ is assigned $\delta_0(v) - 2\alpha$.

When an edge uv is relaxed, reducing by Δ the distance to descendents of v in T, the value of σ is reduced by Δ for all these nodes. Because of the implicit representation, this takes $O(\log n)$ time. At the end of each iteration i, the algorithm repeatedly searches for a rootmost node v^* with $\sigma(v^*) \leq 0$. When it finds such a node v^*, it visits a maximal subtree T' rooted at v^* consisting of nodes v with $\sigma(v) \leq 0$, and, for each such node v, adds a connection (r_i, v) and resets $\sigma(v)$ to $\epsilon\alpha$ if $|T'| \geq \log n$ and to σ(parent of v^*) otherwise. The time to find v^* is $O(\log n)$ and the time to visit T' is $O(|T'|)$. The σ searches continue until there is no node v^* with $\sigma(v^*) \leq 0$, at which point the next iteration of the algorithm commences.

Now for the analysis. Say a σ search is *special* if it leads to resetting $\sigma(v)$ to σ(parent of v^*) for nodes v in v^*'s subtree. For the purpose of the analysis, we place a token on each such node v when this happens. The next time a connection for v is added, we remove that token (though a new one might be placed on v immediately after).

For a vertex v, define $\delta_T(v) = \gamma(\text{root of } T) + \ell(\text{root-to-}v \text{ path in } T)$. At any point during the algorithm's execution, say v is *active* if an edge uv into v was relaxed but since that happened no connection to v has been added to S. Consider the partition defined by connected regions of T with same σ value.

The algorithm maintains the following invariant: (1) For any block of the partition, either the root of the block is active or the block's size $\geq \log n$. (2) If no connection in S involves v then $\sigma(v) = \delta_T(v) - 2\alpha$. (3) If the most recent connection for v is (i^*, v), then the value of $\sigma(v)$ is (a) equal to $\delta_T(v) - \delta_{i^*}(v) + \epsilon\alpha$ if v has no token, or (b) at most that amount if v has a token.

Let $\bar{S}$ be the value of S when the algorithm finishes. The invariant implies that $\bar{S}$ coves all nodes v such that $\min_i \delta_i(v) \leq 2\alpha$. The time for visiting all subtrees T' is $|\bar{S}|$ because each node visited gets a new connection. The number of σ searches that find active nodes v^* is $O(n)$ because the number of activations is the number of relaxation steps. The number of σ searches that find the root of a block of size $\geq \log n$ is $\leq |\bar{S}|/\log n$. The time for all σ searches is thus $O(|S| + n \log n)$.

Each special σ search reduces the number of blocks by one. The number of blocks is initially at most n and increases by at most one per relaxation step. Hence the total number of special σ searches is $O(n)$. Each such search results in $< \log n$ tokens being placed. Thus the total number of tokens placed is $O(n \log n)$.

Finally, we bound $|\bar{S}|$. Define $f(v) = \delta_{i^*}(v)$ where (r_i, v) is the most recent connection for v added to S. Each new connection (r_i, v) reduces $f(v)$ by at least $\epsilon\alpha$ except if it is the first connection for v or a token was removed from v when adding the connection. Since $f(v)$ always lies between 0 and 2α, there are at most $O(\epsilon^{-1})$ nonexceptional connections per node v. The total number of exceptional connections is $O(n \log n)$. Thus $|\bar{S}| = O(n(\epsilon^{-1} + \log n))$.

Acknowledgements

Thanks to Francisco Barahona, Sarah Bell, Maurice Herlihy, Satish Rao, David Reiss, Robert Tarjan, and Mikkel Thorup.

References

[1] S. Alstrup, J. Holm, K. de Lichtenberg, M. Thorup, "Maintaining information in fully-dynamic trees with top trees," ArXiv cs.DS/0310065.

[2] J. R. Driscoll, N. Sarnak, D. D. Sleator, R. E. Tarjan, "Making data structures persistent," *JCSS* 38 (1989), pp. 86-124.

[3] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung, "Maintenance of a minimum spanning forest in a dynamic plane graph," *J. Alg.* 13 (1992), pp. 33-54.

[4] J. Fakcharoenphol and S. Rao, "Planar graphs, negative weight edges, shortest paths, near linear time," *FOCS* (2001), pp. 232-241.

[5] G. N. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications," *SIAM J. Comput.* 16 (1987), pp. 1004-1022.

[6] G. N. Frederickson, "Planar graph decomposition and all pairs shortest paths, *J. ACM* 38 (1991), pp. 162-204.

[7] G. N. Frederickson, "Using cellular graph embeddings in solving all pairs shortest paths problems," *J. Algor.* 19 (1995), pp. 45–85.

[8] G. N. Frederickson, "Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees, *SIAM J. Comput* 26 (1997), pp. 484-538.

[9] R. Hassin, "Maximum flows in (s, t) planar networks," *Inform. Process. Lett.* 13 (1981), pp. 107.

[10] S. Khuller, J. Naor and P. N. Klein, "The lattice structure of flow in planar graphs," *SIAM J. Disc. Math.* 6 (3) pp. 477–490, (1993).

[11] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM J. Appl. Math.* 36 (1979), pp. 177-189.

[12] K. Mehlhorn, S. Näher, "Bounded ordered dictionaries in O(log log N) time and O(n) space," *IPL* 35 (1990), pp. 183-189.

[13] G. L. Miller, "Finding small simple cycle separators for 2-connected planar graphs," *J. Comput. Syst. Sci.* 32 (1986), pp. 265-279.

[14] H. Ripphausen-Lipa, D. Wagner, and K. Weihe, "The vertex-disjoint menger problem in planar graphs," *SIAM J. Comput.* 26 (1997), pp. 331-349.

[15] J. P. Schmidt, "All shortest paths in weighted grid graphs and its application to finding all approximate repeats in strings," *SIAM J. Comput.* 27 (1998), pp. 972-992.

[16] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. System Sci.* 26 (1983), pp. 362-391

[17] M. Thorup, "Compact oracles for reachability and approximate distances in planar digraphs. *FOCS* (2001), pp. 242-251

[18] P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space," *IPL* 6 (1977), pp. 80-82.

[19] K. Weihe, "Maximum (s, t)-flows in planar networks in $O(|V| log |V|)$ time," *JCSS* 55 (1997), pp. 454-476

.