A Subset Spanner for Planar Graphs, with Application to Subset TSP

Philip N. Klein
Brown University
Providence, Rhode Island
USA
klein@cs.brown.edu

ABSTRACT

Let $\epsilon > 0$ be a constant. For any edge-weighted planar graph G and a subset S of nodes of G, there is a subgraph H of G of weight a constant times that of the minimum Steiner tree for S such that distances in H between nodes in S are at most $1+\epsilon$ times the corresponding distances in G. As a consequence, there is an $O(n \log n)$-time approximation scheme for finding a TSP among a given subset of nodes of a planar graph. This is the first PTAS for the problem.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

General Terms

Algorithms

Keywords

approximation scheme,planar graph,traveling salesman problem,spanner

1. INTRODUCTION

A *spanner* of an edge-weighted graph is a subgraph that approximately preserves distances. More specifically, the subgraph is an s-spanner if the distance in the subgraph between any two nodes (with respect to the edge-weights) is at most s times their distance in the original graph. The goal of research in spanners is to determine how small (in some sense) a spanner can be while still approximately preserving distances. The literature concerning spanners is extensive; see [**?**] for a survey.

In this paper we are concerned with the *weight* of a spanner [7, 16], which is defined to be the sum of weights of edges comprising it. Suppose G is a connected graph with finite edge-weights. All distances in G are finite, so a spanner of G must at the very least achieve connectivity. This shows [7] that the weight of a spanner for G is at least the weight $MST(G)$ of G's minimum spanning tree.

STOC'06, May21–23, 2006, Seattle, Washington, USA.

1.1 Previously known spanner result for planar graphs

Althöfer, Das, Dobkin, Joseph, and Soares [3] proved the following spanner result for planar graphs:

THEOREM 1.1 (ALTHÖFFER ET AL.). *For any planar graph G with edge-weights and any $\epsilon > 0$, there is a $(1+\epsilon)$-spanner of weight at most $(1 + 2\epsilon^{-1})\ MST(G)$*

Their proof is constructive, and gives a polynomial-time algorithm to construct the promised spanner. A linear-time algorithm was given recently [15].

1.2 Use of spanners in TSP approximation

The *metric TSP* problem is as follows: given a metric space (a set of nodes with symmetric pairwise distances satisfying the triangle inequality), find a *tour* (a closed path that visits each node once) of minimum total distance. Any undirected graph with edge-weights defines a metric space on its nodes (the distance between two nodes is defined to be the minimum weight of a path between the nodes), so each graph defines an instance of metric TSP. A tour then corresponds in the original graph to a closed walk that is allowed to traverse edges more than once; the weight of an edge contributes to the total weight of the tour according to its multiplicity.

The general problem of metric TSP is MAX-SNP-hard [19] but one can obtain polynomial-time approximation schemes by restricting attention to special kinds of metrics. For undirected planar graphs with all edge-weights equal to 1, Grigni, Koutsoupias, and Papadimitriou [11] gave an approximation scheme: for every $\epsilon > 0$, there is an algorithm that takes time $O(n^{\epsilon^{-1}})$ and that finds a tour of weight at most $1 + \epsilon$ times optimal.

For undirected planar graphs with arbitrary edge-weights, Arora, Grigni, Karger, Klein, and Woloszyn [5] gave an approximation scheme that takes time $O(n^{\epsilon^{-2}})$. The first step of this algorithm is to find a $(1 + \epsilon/2)$-spanner of the planar input graph, using the construction of Althöffer et al.

For the same problem, Klein [15] has given an approximation scheme that takes time $O(c^{\epsilon^{-2}} n)$ for a constant c. Thus the algorithm takes $O(n)$ time for fixed $\epsilon > 0$. Like

the algorithm of Arora et al., this algorithm's first step is to find a $(1+\epsilon/2)$-spanner of the input graph.

For *Euclidean metrics* in finite dimension, Arora [4] and Mitchell [17] gave approximation schemes. Building on the work of Arora, Rao and Smith [20] gave a faster approximation scheme for Euclidean metrics; for the case of two-dimensions, their algorithm takes time $O(\epsilon^{-O(\epsilon)}n + n\log n)$. The algorithm of Rao and Smith employed a spanner result, due to Das, Narasimhan, and Salowe [8], for the complete Euclidean graph on a set of points. This result showed that a $(1+\epsilon)$-spanner could be found whose weight was $O(a_d(\epsilon))$ times the weight of the minimum spanning tree. Here $a_d(\epsilon)$ is a function only of the dimension d and the precision parameter ϵ.

1.3 TSP in a submetric space of that defined by an edge-weighted planar graph

Imagine a delivery-truck driver is looking over a city road map, and wants to find the least time-consuming route that visits a specified set of locations. Distances are of course not Euclidean because shortest paths follow roads and bridges (which themselves might have different durations). The road map is (mostly) planar, but the planar TSP approximation scheme does not apply because it requires the tour to visit *all* nodes. A more natural problem in this context is that of finding a tour that visits a given subset S of nodes.

The goal, equivalently, is to find a TSP in a *submetric space* of the metric space defined by a planar graph, namely the submetric space induced by the given set of nodes. This problem, in addition to being a natural problem in road maps, is a generalization both of TSP in a planar graph and TSP in the Euclidean plane, as observed by Arora et al. [5], who gave a *quasipolynomial-time approximation scheme*. To find a tour of weight at most $1+\epsilon$ times optimal, their algorithm requires time $O(n^{\mathrm{poly}(\log n, 1/\epsilon)})$. They make the following conjecture.

CONJECTURE 1.2 (ARORA ET AL.). *There exists a function $f(\cdot)$ such that: given $\epsilon > 0$, a planar graph G with edge-weights, and a subset S of vertices, there exists an edge-induced subgraph G' that $(1+\epsilon)$-approximates all distances between nodes in S, and furthermore G' has total edge weight at most $f(\epsilon)$ times the minimum Steiner tree weight for S.*

Note that the minimum Steiner tree weight for S can be arbitrarily less than the minimum spanning tree weight.

Arora et al. observed that their approximation scheme for weighted planar TSP could be extended to the subset TSP if the conjecture were true (and if the subgraph could be found in polynomial time). In fact, the same holds for the linear-time approximation scheme of [15], as we indicate in the next section.

1.4 The subset spanner for weighted planar graphs

In this paper, we prove the conjecture with $f(\epsilon) = O(\epsilon^{-4})$. The proof is constructive, and implies a polynomial-time algorithm to construct a spanner. We also describe an efficient construction algorithm, one that takes $O(n\log n)$ time for an n-node graph and fixed ϵ.

THEOREM 1.3. *There is an algorithm that, given $\epsilon > 0$, a planar graph G with edge-weights, and a node-subset S, selects an edge-induced subgraph G' with the following properties:*

- *G' has weight $O(\epsilon^{-4})$ times the weight of the minimum-weight tree in G that spans all nodes in S.*
- *for every pair of nodes $u, v \in S$, the u-to-v distance in G' is at most $1+\epsilon$ times the u-to-v distance in G.*

The algorithm takes $O(\epsilon^{-1} \cdot n\log n)$ time.

The TSP approximation scheme of [15] consists of (1) using a spanner result to extract a subgraph from the input graph, (2) using a "shifting" technique to choose some edges to contract, (3) using dynamic programming to find an optimal tour in the contracted graph, and (4) transforming the optimal tour in the contracted graph into a tour in the uncontracted graph by incorporating copies of the contracted edges. The dynamic program in Step 3 can easily be modified to find an optimal tour visiting nodes in S. Steps 2 and 4 remain unchanged. We need only use the new spanner result in Step 1 to complete the approximation scheme for subset TSP.[1] We therefore obtain the following corollary.

COROLLARY 1.4. *For any fixed $\epsilon > 0$, there is an $O(n\log n)$ algorithm that, given a planar graph with edge-weights and a subset S of nodes, finds a walk that visits all nodes in S and that has weight at most $1+\epsilon$ times optimal.*

This is the first PTAS for the problem.

In Section 2, we give some graph terminology and notation. In Section 3, we outline a basic charging technique used in the construction. The construction is outlined in the subsequent sections. At the end of each of these sections, we make some remarks concerning efficient implementation. Readers interested only in the existence of subset spanners can skip these remarks.

2. TERMINOLOGY AND NOTATION

In this paper, we focus on undirected graphs. Each undirected edge corresponds to two directed *darts*, one in each direction. A dart is considered to point from its tail to its head. A *walk* is a sequence $d_1\ \ldots\ d_k$ of darts such that the head of d_i is the tail of d_{i+1}, for $i = 1, \ldots, k-1$. It is a *closed* walk if the head of d_k is the tail of d_1. It is a *path* if every edge is represented in the path by at most one dart. A *cycle* is a path that is a closed walk. A cycle $d_1\ \ldots\ d_k$ is *simple* if every node occurs at most once as the head of a dart d_i. Similarly, a path $d_1\ \ldots\ d_k$ is *simple* if it is not closed and every node occurs at most once as the head of a dart d_i. For a walk $W = d_1, \ldots, d_k$, the set of edges represented is denoted by $E(W)$. We denote by $\mathrm{start}(W)$ the tail of d_1, and we denote by $\mathrm{end}(W)$ the head of d_k. We denote the reverse of W by $\mathrm{rev}(W)$. We use $\circ$ to denote concatenation of paths. We say a node belongs to W if the node is the head or tail of one of the darts comprising W. If nodes x and y belong to a simple path P, we denote by $P[x,y]$ the subwalk of P connecting x and y. We denote by $\mathrm{dist}_G(x,y)$ the minimum weight of an x-to-y path in G, and we omit the subscript when no ambiguity would result. Suppose $d_0\ \ldots\ d_{k-1}$ is a simple cycle C, and x is the tail of d_i and y is the head of d_j. We then denote by $C[x,y]$ the simple path $d_i\ d_{i+1 \pmod k}\ d_{i+2 \pmod k}\ \cdots\ d_j$.

The graphs arising in this paper are assumed without loss of generality to be biconnected. All weight assignments are

[1]The running time of the approximation scheme is $O(\epsilon^{-1} \cdot n\log n + c^{\epsilon^{-5}} n)$.

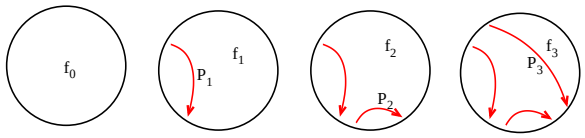

Figure 1: This figure illustrates the situation addressed by the basic charging scheme. Starting with a face f_0, paths are added one by one. Path P_i divides face f_{i-1} into two faces, f_i and g_i. Traversing the boundary of f_i counterclockwise traverses P_i in the forward direction.

assumed to be nonnegative. A *planar embedded graph* (or *plane graph*) is one embedded in the plane with no edge-crossings. Faces are defined in the usual way. Assuming biconnectivity, the boundary of every face is a simple cycle. We identify a face with the counterclockwise-oriented version of this cycle. Assuming the graph is connected, there is exactly one *infinite face*. The *boundary* of a plane graph is the counterclockwise walk along the boundary of the infinite face. A node/edge is *enclosed* by a simple cycle in a plane graph if the node/edge is embedded inside the cycle or is on the cycle. It is *strictly enclosed* if it is enclosed but not on the cycle.

For a cycle C embedded in the plane, if x and y are on C, we denote by $C[x, y]$ the x-to-y subpath of C that goes counterclockwise around C.

A connected planar embedded graph defines another connected, planar embedded graph, the *dual graph*. There is a bijection between the edges of the dual graph and the edges of the original (*primal*) graph. A classical result in planarity states that if T is a spanning tree of a planar embedded graph then the set of dual edges *not* corresponding to edges of T forms a spanning tree of the dual graph.

3. BASIC CHARGING TECHNIQUE

Let G be a planar embedded graph with nonnegative edge-weights $\ell(\cdot)$. Let H_0 be a subgraph of G, let f_0 be a face of H_0, and let $P_1, P_2, \ldots, P_k$ be a set of edge-disjoint paths in G that are edge-disjoint from H_0. (See Figure 1.) For $i = 1, 2, \ldots, k$, let H_i denote the subgraph of G formed by $H_0, P_1, P_2, \ldots, P_i$. We assume that, for $i = 1, 2, \ldots, k$,

- the path P_i shares only its endpoints with H_{i-1},
- the edges of P_i are strictly enclosed by a face f_{i-1} of H_{i-1}, splitting f_{i-1} into two faces, g_i and f_i, in H_i, where

$$g_i = f_{i-1}[\text{start}(P_i), \text{end}(P_i)] \circ \text{rev}(P_i)$$

and

$$f_i = P_i \circ f_{i-1}[\text{end}(P_i), \text{start}(P_i)]$$

- $(1+\epsilon)\ \ell(P_i) < \ell(f_{i-1}[\text{start}(P_i), \text{end}(P_i)])$

LEMMA 3.1.

$$\ell(P_1 \cup \cdots \cup P_k) < \epsilon^{-1}\ell(f_0)$$

PROOF. For $i = 1, \ldots, k$,

$$\begin{aligned}\ell(f_i) &= \ell(f_{i-1}) - \ell(f_{i-1}[\text{start}(P_i), \text{end}(P_i)]) + \ell(P_i) \\ &< \ell(f_{i-1}) - \epsilon\ \ell(P_i)\end{aligned}$$

This recurrence shows $\ell(f_k) < \ell(f_0) - \epsilon \sum_i \ell(P_i)$, which proves the lemma. □

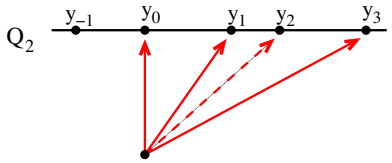

Figure 2: This diagram illustrates the process of constructing a single-source spanner. The shortest paths to the nodes y_i are considered one at a time, in order. Such a path is added to the spanner only if it is much shorter than the indirect path that uses the previously added path followed by a subpath of Q_2

4. SINGLE-SOURCE SPANNER

First we give a construction for a single-source $(1+\epsilon)$-spanner. The technique is adapted from [13, 22], though that work does not address weight, only cardinality.

THEOREM 4.1. *Let G be a plane graph with nonnegative edge-weights $\ell(\cdot)$, let Q be a shortest path in G between its endpoints, and let R be a shortest path from some node x to Q. Then for any $\epsilon > 0$ there is a subgraph H of G of weight $< 4(\epsilon^{-1} + \epsilon^{-2})\ell(R)$ such that, for each node y of Q,*

$$\text{dist}_{H \cup Q \cup R}(x, y) \le (1+\epsilon)\ \text{dist}_G(x, y)$$

and such that H consists of $O(\epsilon^{-1})$ shortest paths from x to some nodes of Q.

PROOF. Let y_0 be end(R), the node of Q closest to x, and number the nodes of Q as $\ldots, y_{-3}, y_{-2}, y_{-1}, y_0, y_1, y_2, y_3, \ldots$. We find the shortest paths comprising H in two passes, one involving shortest paths to nodes $y_1, y_2, y_3, \ldots$, and one involving shortest paths to nodes $y_{-1}, y_{-2}, y_{-3}, \ldots$. We describe and analyze the first pass; the other is symmetric.

```
initialize j := 0 and k := 0
for i := 1, 2, ...,
  repeat
    j := j + 1
    if there is no node y_j
      or ℓ(Q[y_0, y_j]) > ε^{-1} ℓ(R)
      then exit
  until (1 + ε) dist(x, y_j) < dist(x, y_k) + ℓ(Q[y_k, y_j])
  P_i := shortest x-to-y_j path
  k := j
```

Let I be the final value of i. Let J be the final value of j. Let f_0 be the face $R \circ Q[y_0, y_{J-1}] \circ$ shortest x-to-y_{J-1} path. We apply the basic charging technique of Section 3 to the paths $P_1, \ldots, P_{I-1}$.

By the stopping condition, $\ell(Q[y_0, y_{J-1}]) \le \epsilon^{-1}\ell(R)$. By the triangle inequality, therefore, $\text{dist}(x, y_{J-1}) \le (1+\epsilon^{-1})\ \ell(R)$, so $\ell(f_0) \le 2(1+\epsilon^{-1})\ \ell(R)$, so by Lemma 3.1, $\ell(P_1 \cup \cdots \cup P_{I-1}) < \epsilon^{-1} \cdot 2(1+\epsilon^{-1})\ \ell(R)$. This shows the weight bound.

The analysis of the number $I - 1$ of paths is adapted from [13, 22]. Let $P_0 = R$.

$$\begin{aligned}\ell(P_i) &\le (1+\epsilon)^{-1}\ (\ell(P_{i-1}) + \ell(Q[\text{end}(P_{i-1}), \text{end}(P_i)])) \\ &\le (1+\epsilon)^{-1}\ \ell(P_{i-1}) + \ell(Q[\text{end}(P_{i-1}), \text{end}(P_i)]) \\ &\le \ell(P_{i-1}) - (\epsilon - \epsilon^2)\ \ell(P_{i-1}) + \ell(Q[\text{end}(P_{i-1}), \text{end}(P_i)]) \\ &\le \ell(P_{i-1}) - (\epsilon - \epsilon^2)\ \ell(R) + \ell(Q[\text{end}(P_{i-1}), \text{end}(P_i)])\end{aligned}$$

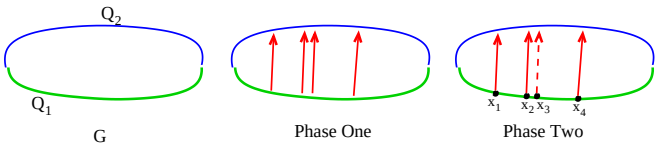

Figure 3: This diagram illustrates part of the bipartite-spanner construction. The boundary of the graph is a cycle consisting of two paths, Q_1 and Q_2. Phase One finds for each node x of Q_1 the distance to Q_2. Phase Two considers the nodes of Q_1 in left-to-right order. If x_i is far along Q_1 from x_{i-1} relative to its distance from Q_2, then x_i is included in S.

because R is a shortest x-to-Q path. Subtracting $\ell(Q[y_0, \text{end}(P_i)])$ from both sides, we obtain

$$\begin{aligned}&\ell(P_i) - \ell(Q[y_0, \text{end}(P_i)])\\ &\quad\leq\quad \ell(P_{i-1}) - \ell(Q[y_0, \text{end}(P_{i-1})]) - (\epsilon - \epsilon^2)\ \ell(R).\end{aligned}$$

This recurrence shows

$$\begin{aligned}&\ell(P_{I-1}) - \ell(Q[y_0, \text{end}(P_{I-1})])\\ &\quad\leq\quad \ell(R) - (I-1)(\epsilon - \epsilon^2)\ \ell(R) \qquad (1)\end{aligned}$$

Since $Q[y_0, \text{end}(P_{I-1})$ is a shortest path, $\ell(Q[y_0, \text{end}(P_{I-1})]) \leq \ell(R) + \ell(P_{I-1})$ by the triangle inequality. This implies

$$\ell(P_{I-1}) - \ell(Q[y_0, \text{end}(P_{I-1})]) \geq -\ell(R)$$

which, together with (1) shows that $I - 1 \leq 2(\epsilon - \epsilon^2)^{-1}$. □

5. BIPARTITE SPANNER

THEOREM 5.1. *Let G be a plane graph with nonnegative edge-weights $\ell(\cdot)$ whose boundary is a cycle formed by two paths, Q_1 and Q_2, where Q_2 is a shortest path. Then for any $\epsilon > 0$ there is a subgraph H of weight at most $c\epsilon^{-3} \cdot \ell(Q_1)$ such that, for each node x in Q_1 and each node y in Q_2,*

$$\text{dist}_{H \cup Q_2}(x, y) \leq (1 + O(\epsilon))\ \text{dist}_G(x, y) \qquad (2)$$

where c is a constant.

The construction of H consists of three phases

Phase One: For each node x of Q_1, let $F[x]$ denote a shortest x-to-Q_2 path.

Phase Two: Let $x_0, x_1, x_2, \ldots, x_s$ be the nodes of Q_1 in left-to-right order. Compute a subset S of these nodes as follows.

```
initialize S := ∅
initialize i := 0
for j := 1, 2, ..., s − 1,
  if ℓ(Q1[xi, xj]) > ε · ℓ(F[xj])
    S := S ∪ {xj}
    i := j
```

Phase Three:

For each node x of S, apply Theorem 4.1 with $Q = Q_2$ and $R = F[x]$, obtaining a single-source spanner of weight $O(\epsilon^{-2}\ \ell(F[x]))$, not including the weight of Q_2. Let H be the union of the single-source spanners, together with Q_1.

Now we bound the weight of H.

LEMMA 5.2. $\sum_{x \in S} \ell(F[x]) \leq \epsilon^{-1}\ \ell(Q_1)$

PROOF. Let $j_1 < j_2 < \cdots < j_t$ be the values of j for which x_j was added to S. Then

$$\begin{aligned}\sum_{k=1}^{t} \ell(F[x_{j_k}]) &\leq \epsilon^{-1} \sum_{k=1}^{j} \ell(Q_1[x_{j_{k-1}}, x_{j_k}])\\ &\leq \epsilon^{-1}\ \ell(Q_1).\end{aligned}$$

□

Lemma 5.2 implies that $\ell(H) = O(\epsilon^{-3} \cdot \ell(Q_1))$.

Now we show that (2) holds for any nodes x and y of Q_1 and Q_2, respectively. The algorithm for Phase Two ensures that there is a node $x' \in S$ such that $\ell(Q_1[x, x']) \leq \epsilon\ \ell(F[x])$. By choice of $F[x]$,

$$\ell(F[x]) \leq \text{dist}_G(x, y)$$

so

$$\ell(Q_1[x, x']) \leq \epsilon\ \text{dist}_G(x, y). \qquad (3)$$

By the triangle inequality,

$$\begin{aligned}\text{dist}_G(x', y) &\leq \text{dist}_G(x, y) + \ell(Q_1[x, x'])\\ &\leq \text{dist}_G(x, y) + \epsilon\ \text{dist}_G(x, y).\end{aligned}$$

Since $x' \in S$, H includes a single-source spanner for x', so

$$\begin{aligned}\text{dist}_H(x', y) &\leq (1 + \epsilon)\ \text{dist}_G(x', y)\\ &\leq (1 + \epsilon)\ (1 + \epsilon)\ \text{dist}_G(x, y).\end{aligned}$$

By the triangle inequality,

$$\begin{aligned}\text{dist}_H(x, y) &\leq \text{dist}_H(x', y) + \ell(Q_1[x, x'])\\ &\leq (1 + \epsilon)\ (1 + \epsilon)\ \text{dist}_G(x, y) + \epsilon\ \text{dist}_G(x, y)\end{aligned}$$

which proves (2). This completes the proof of Theorem 5.1.

Efficient construction

We now discuss the efficient construction of the bipartite spanner, including construction of single-source spanners for a subset of the nodes of Q_1. We need two algorithmic tools.

Dynamic-tree data structure

A dynamic-tree data structure (versions include ST-trees/link-cut trees [21], linear attribute grammars [6], topology trees [10], top trees [2], RC-trees [1], self-adjusting top trees [23]) represents a forest of disjoint trees under topological operations and label operations. Each operation takes $O(\log n)$ amortized time.

As described in [23], the data structure can maintain planar embeddings of the trees as well. This feature will be useful to us; the trees will be spanning trees of plane graphs, and so will inherit their embeddings from the plane graphs.

For the purposes of this paper, we need to represent *rooted* trees under two topological operations, *cut*, which removes a given edge, breaking a tree into two trees, and *link*, which adds an edge from the root of one tree to a node of another, combining them into two trees.

The data structure is also capable of maintaining labelings of the nodes/edges under various operations. Assume the labels for some labeling are integers. Implementable operations include:

- adding/multiplying all the nodes/edges in a path/rooted-subtree by a constant;
- finding the node/edge with the minimum label;
- finding the leafmost/rootmost/leftmost/rightmost[2] node/edge having a negative label.

The multiple-source algorithm

Consider a directed graph and a rooted spanning tree T directed from root to leaves. If a and b are arcs with the same head v, and T contains a, we refer to the operation of removing a from T and inserting b as a *pivot replacing a with b reducing distance by δ*, where

$$\begin{aligned}\delta &= \text{distance to } v \text{ in tree before pivot} \\ &\quad -\text{distance to } v \text{ in tree after pivot}\end{aligned}$$

Suppose T_1 is rooted at v and T_2 is rooted at u, and uv is an arc. Then T_2 can be obtained from T_1 by

1. removing the arc of T entering u and adding the arc uv (rerooting the tree at u), and
2. performing a sequence of pivots.

Let $\Delta(T_1, T_2)$ be the set of pivots in Step 2.

Klein [14] has given a multiple-source shortest-paths algorithm for planar directed graphs. The input is a planar embedded directed graph with nonnegative arc-lengths whose boundary nodes are $r_1, r_2, \ldots, r_k$ in counterclockwise order around the boundary of the graph. The algorithm produces, in sequence, the shortest-path trees rooted at $r_1, r_2, \ldots, r_k$. The algorithm starts by constructing an r_1-rooted shortest-path tree. Then the algorithm reroots the tree at r_2, and performs a series of pivots to obtain an r_2-rooted shortest-path tree. It then reroots the tree at r_3, and so on, ending with the r_k-rooted shortest-path tree. In this way, the algorithm finds the pivot sets $\Delta(T_1, T_2), \Delta(T_2, T_3), \ldots, \Delta(T_{k-1}, T_k)$. The algorithm takes $O(n \log n)$ time on n-node graphs.

In order to select the pivots, the multiple-source algorithm maintains a dynamic-tree representation of the dual spanning tree consisting of edges not corresponding to edges of T. Choosing the pivot to perform consists in finding a leafmost edge with a negative label. Carrying out the pivot involves two steps, changing the topology of the primal tree and reducing by some number δ the distance to a node and its descendents in the tree. The first step is implemented by two topological operations on the dual tree, and the second step is implemented by two label operations in each of which the labels on a path are modified.

Bipartite spanner algorithm

Now we describe an $O(n \log n)$ algorithm for constructing the spanner described in Theorem 5.1. (This bound assumes ϵ is a constant.)

The input is a graph G with edge-weights $\ell(\cdot)$ whose boundary consists of two paths, Q_1 and Q_2. For Phases One and Two, we need to compute for each node x of Q_1 the distance $d(x)$ to Q_2. Define a new weight assignment that is identical to $\ell(\cdot)$ except that the edges of Q_2 are assigned weight zero, and compute shortest-path distances $d(x)$ from an arbitrary node of Q_2 with respect to the modified weight assignment.

For Phase Three, we need to compute the union of single-source spanners for a set S of nodes of Q_1. Let $x_1, \ldots, x_s$ and $y_1, \ldots, y_t$ be the nodes of Q_1 and Q_2, respectively, in order from left to right. (See Figure 4.) The idea is to first run the multiple-source shortest-path algorithm to find the pivots needed to go from root x_1 to root x_2 and so on, up to root x_s. Then the shortest-path tree is represented by a dynamic tree, enabling the algorithm to find, successively, the single-source spanner for x_1, then for x_2, and so on.

The dynamic tree maintains three node-labelings:

- the labeling L_1 that gives the distance of each node in the current shortest-path tree,
- a node-labeling L_2 to help identify nodes in Q_2 to which to add shortest paths, and
- a 0-1 edge-labeling L_3 to keep track of edges belonging to shortest x-to-y_i paths that have been selected for inclusion.

For two nodes y_i and y_j, define $q(y_i, y_j) = \ell(Q[y_1, y_j]) - \ell(Q[y_1, y_i])$. Note that $q(y_i, y_j) + q(y_j, y_k) = q(y_i, y_k)$.

The following algorithm implements one pass of the single-source algorithm for each node $x_i \in S$. At the end of the algorithm, the edges assigned 0 by labeling L_3 are the edges that belong to the spanner according to that pass. The algorithm for the other pass is symmetric.

1. Initialize T to be the shortest-path tree of G rooted at x_1.
2. Initialize labels L_1 so that $L_1(v) = x_1$-to-v distance
3. Initialize labels L_2 so that, for each y_j,
$$L_2(y_j) = (1+\epsilon)\ L_1(y_j) - q(y_1, y_j) \qquad (4)$$
4. Initialize labels L_3 to be all 1.
5. for $i = 1, \ldots, s$
   (a) *Comment:* T is the x_i-rooted shortest-path tree, and $L_1(v)$ is the x_i-to-v distance.
   (b) If x_i is not in S then go to Step 5i
   (c) Initialize k so that y_k is the node of Q_2 closest to x_i.
   (d) Add $q(y_1, y_k) - L_1(y_k)$ to all L_2 labels.
   (e) *Comment:* For every node y_j,
   $$L_2(y_j) = (1+\epsilon)\ L_1(y_j) - L_1(y_k) - q(y_j, y_j) \quad (5)$$
   (f) while there is a node y_j with $j > k$ such that $L_2(y_j) < 0$,
      i. Let y_{j^*} be the lowest-numbered such node.
      ii. Assign 0 to all edges on the y_{j^*}-to-x_i path.
      iii. Add $\ell(Q_2[y_k, y_{j^*}]) + L_1(y_j) - L_1(y_{j^*})$ to all L_2 labels.
      iv. Let $k = j$.
      v. *Comment:* Now (5) holds again.
   (g) Add $L_1(y_k) - \ell(Q_2[y_1, y_k])$ to all L_2 labels.
   (h) *Comment:* Now (4) holds again.
   (i) If $i < s$, perform topological operations as dictated by the multiple-source algorithm to transform T to a shortest path tree rooted at x_{i+1}. Perform corresponding operations on L_1 to achieve $L_1(v) = x_{i+1}$-to-v distance, and operations on L_2 to re-establish (4).

[2] with respect to the planar embedding

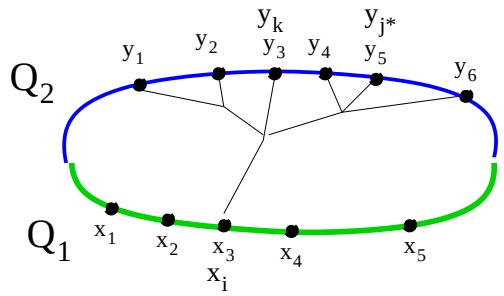

Figure 4: This figure illustrates a moment in the bipartite-spanner algorithm. T is a shortest-path tree rooted at x_3. The last shortest path included in the spanner is the path to y_3, and the next one to be added will be the one to y_5.

By (5), the while-loop mimics the single-source spanner construction; this shows correctness of the algorithm. Now we consider the running time. Step 5c, 5d, 5(f)i, 5(f)ii, 5(f)iii, and 5g can be implemented by dynamic-tree label operations. Therefore

- each iteration of the while-loop of Step 5f takes amortized $O(\log n)$ time, and
- each iteration of the for-loop in Step 5 takes $O(\log n)$ time, not counting the execution of the while-loop.

During iteration i of the for-loop, each iteration of the while-loop adds a shortest path to the single-source spanner for x_i, so Theorem 4.1 implies that the number of while-loop iterations is $O(\epsilon^{-1})$. It follows that the algorithm above takes $O(\epsilon^{-1} n \log n)$ time, where n is the number of nodes in G.

6. BOUNDARY-TO-BOUNDARY SPANNER

THEOREM 6.1. *Let G be a plane graph with nonnegative edge-weights $\ell(\cdot)$. Let C be the boundary of G. Then for any $\epsilon > 0$ there is a subgraph $\hat{H}$ of weight $O(\epsilon^{-4} \cdot \ell(C))$ such that, for any nodes x and y in C_0,*

$$\text{dist}_{\hat{H}}(x, y) \leq (1+\epsilon)\text{dist}_G(x, y)$$

PROOF. To prove the theorem, we give a recursive algorithm. The recursive step is illustrated in Figure 5.

BOUNDARYSPANNER(G):

1. Let C be the boundary of G.
2. Consider the pairs of nodes x, y such that
$$(1+\epsilon)\ \text{dist}_G(x, y) < \ell(C[x, y]) \tag{6}$$
3. If there are no such nodes x and y then return C.
4. Let $\hat{x}$ and $\hat{y}$ be nodes such that (6) holds for $x = \hat{x}$ and $y = \hat{y}$ but does not hold for any other pair x, y of nodes lying on $C[\hat{x}, \hat{y}]$.
5. Let B be the path $C[x, y]$, and let P be a shortest x-to-y path.
6. Let L be the subgraph of G consisting of nodes and edges enclosed by the cycle $B \circ \text{rev}(P)$.

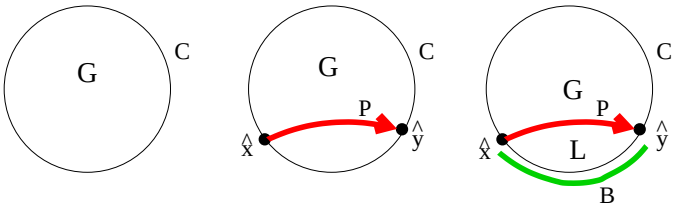

Figure 5: This figure illustrates the recursive case for the boundary-to-boundary-spanner construction. Choose a shortest path P from a boundary node $\hat{x}$ to another boundary node $\hat{y}$. The counterclockwise portion of the boundary from $\hat{x}$ to $\hat{y}$ is B. The subgraph bounded by B and P is called L. The construction finds a bipartite spanner for L and adds it to a recursive boundary-to-boundary spanner for the rest of the graph.

7. Let G' be the subgraph of G consisting of nodes and edges enclosed by the cycle $P \circ C[y, x]$
8. Using the construction of Theorem 5.1, obtain a subgraph H of L having weight at most $d\epsilon^{-3}\ \ell(B)$ such that, for each node x in B and each node y in P,
$$\text{dist}_{H \cup P}(x, y) \leq (1+\epsilon)\text{dist}_L(x, y) \tag{7}$$
where d is a constant.
9. Return the subgraph consisting of H together with BOUNDARYSPANNER(G').

The algorithm is called on the input graph to obtain the subgraph $\hat{H}$. Suppose the last recursive call (when the base case applies) is the r^{th}. For $i = 1, 2, \ldots, r$, let $G_i, C_i, B_i, P_i, L_i, H_i$ denote the values of G, C, B, P, L, H respectively in the i^{th} call. The base case returns C_r.

First we prove the weight bound. Each edge of $\bigcup_i C_i$ occurs in exactly one of $C_r, B_1, \ldots, B_{r-1}$. Hence

$$\ell(C_r) + \sum_i \ell(B_i) = \ell(\bigcup_i C_i). \tag{8}$$

Note that $\bigcup_i C_i = C_1 \cup \bigcup_i P_i$. Hence

$$\ell(\bigcup_i C_i) = \ell(C_1 \cup \bigcup_i P_i). \tag{9}$$

By Lemma 3.1, $\sum_i \ell(P_i) \leq \epsilon^{-1}\ \ell(C_1)$. Combining this with (8) and (9), we get

$$\ell(C_r) + \sum_i \ell(B_i) \leq (1+\epsilon^{-1})\ \ell(C_1). \tag{10}$$

Let H_i be the subgraph H obtained in Step 8 in the i^{th} recursive call. Then

$$\begin{aligned} \sum_i \ell(H_i) &\leq d\epsilon^{-3} \sum_i \ell(B_i) \\ &\leq d(\epsilon^{-4} + \epsilon^{-3})\ \ell(C_1) \end{aligned}$$

which proves the bound stated in the theorem.

Now we prove the distance-preserving property. Let R be a shortest path in G_1 whose endpoints lie on C_1. Since the paths P_i are shortest paths, we can assume without loss of generality that R never crosses a path P_i twice. Hence R can be written as $R_1 \circ R_2 \circ \cdots \circ R_t$, where each R_j lies

entirely in some subgraph L_i. The start and end of R_j must be boundary nodes of L_i, i.e. must belong to $B_i \cup P_i$. We replace R_j with an alternative path R'_j in $H_i \cup P_i$. Let x, y be the endpoints of R_j.

Case I: x and y both belong to P_i. Since P_i is a shortest path in L_i, the subpath of P_i between the endpoints of R_j is a shortest path in L_i. In this case, we let R'_j be this subpath.

Case II: Case I does not hold, and x and y both belong to B_i. Since Case I does not hold, x and y are not the endpoints of P_i, so by choice of the nodes $\hat{x}$ and $\hat{y}$ in Step 4, $(1+\epsilon)\ \mathrm{dist}_{G_i}(x,y) > \ell(B_i[x,y])$. In this case, we let R'_j be $B_i[x,y]$.

Case III: Cases I and II do not hold. In this case, x (say) is on B_i and y is on P_i. For this case, by (7),

$$\mathrm{dist}_{H_i \cup P_i}(x,y) \le (1+\epsilon)\ \mathrm{dist}_{L_i}(x,y)$$

so we let R'_j be the shortest x-to-y path in $H_i \cup P_i$.

In each case, R_j is replaced with a path R'_j such that $\ell(R'_j) \le (1+\epsilon)\ \ell(R_j)$. Combining the replacement paths yields a path R' such that $\ell(R') \le (1+\epsilon)\ \ell(R)$, proving the distance-preserving property of the theorem. □

Efficient construction

We now describe an $O(n \log n)$ algorithm for constructing the spanner described in Theorem 6.1.

Although the boundary is a cycle, we conceptually break it at a node x_1: we avoid choosing an x-to-y path P such that the counterclockwise x-to-y subpath of the boundary includes x_1. Consequently, x_1 remains a boundary node throughout the algorithm. For boundary nodes x and y, we say y is *counterclockwise from* x if the counterclockwise x-to-y subpath of the boundary does not include x_1.

Like the algorithm for the bipartite spanner, this construction algorithm employs the multiple-source shortest-path algorithm [14]. However, in this case steps of the construction algorithm are intermixed with steps of the multiple-source shortest-path algorithm.

The algorithm maintains a tree that is shortest not with respect to the original edge-weights, but with respect to modified edge-weights in which each edge's weight has been increased by a factor of $1+\epsilon$ except for those edges on the boundary. (The nonboundary edges pay a percentage ϵ "tax".) Initially all edge-weights are the taxed weights; then, one by one, each boundary edge's weight is decreased (the tax is waived). Subsequently, whenever an edge becomes a boundary edge, the tax on that edge is waived.

EfficientBoundarySpanner(G):
initialize weights of all edges to be the taxed weights.
for each boundary edge, waive the tax.
let $x_1, \ldots, x_s$ be boundary nodes in counterclockwise order.
initialize T to be the x_1-rooted shortest-path tree.
for $j := 2$ to s
  reroot T at x_j
  perform pivots to transform T to shortest-path tree.
  repeat
1 find maximal tree-path of the form $x_j x_{j-1} \ldots x_{k+1} x_k$
2 if $k = 1$ then skip to the next for-loop iteration.
3 let P be the tree path from x_{k-1} to the root.
4 let B be the x_{k-1}-to-x_j subpath of the boundary.
5 let L be the graph consisting of nodes and edges enclosed by $B \circ \mathrm{rev}(P)$
6 carry out Step 8 of BoundarySpanner
7 delete edges and nodes in L except for those on P.
8 waive the tax on edges of P.

The algorithm EfficientBoundarySpanner maintains the following invariant:

> If, in iteration j of the for-loop, T contains an x_p-to-x_q path (where $q < p$) consisting of nonboundary edges then $p = j$.

By choice of the maximal tree-path in Step 1, the x_{k-1}-to-root path P found in Step 3 does not consist entirely of boundary edges. If the maximal prefix of P consisting of nonboundary edges were not P itself, it would violate the invariant. Consequently, the path P found in Step 3 consists entirely of nonboundary edges.

Since P is a shortest path, the length of P is less than the weight of the boundary path $x_k x_{k-1} \ldots x_j$. Because of taxation, the length of P is $1+\epsilon$ times $\ell(P)$, so inequality (6) is satisfied with $x = x_{k-1}$ and $y = x_j$. Moreover, for any $x, y \in \{x_k, x_{k+1}, \ldots, x_j\}$, the boundary path from x to y is a shortest path, so inequality (6) does not hold. Thus x_{k-1} and x_j can play the role of $\hat{x}$ and $\hat{y}$ in Step 5 of the procedure BoundarySpanner.

The deletions of Step 7 of EfficientBoundarySpanner result in the graph G' of Step 7 of BoundarySpanner. Thus EfficientBoundarySpanner correctly implements BoundarySpanner. Because the repeat-loop continues until $x_1 x_2 \ldots x_j$ is a shortest path, the invariant is preserved.

Now we address the running time. In Step 1, the tree-path of boundary edges can be found by just tracing down the tree, always choosing the leftmost child. Each of the edges in this path will be deleted in Step 6, so the total time for all executions of Step 1 is linear. Similarly, in Step 3, the tree path can be found by tracing up the tree. By the invariant, none of these edges are boundary edges, and, after Step 6, they all become boundary edges, so the total time for Step 3 is linear.

Similarly, each edge is deleted in Step 7 at most once, and each edge has its tax waived, at most once in Step 8. Thus there is a linear number of such operations over the entire execution of EfficientBoundarySpanner. Recall that the multiple-source shortest-path algorithm maintains a dynamic-tree representation of the dual spanning tree. Each deletion corresponds to the contraction of an edge in the dual spanning tree. Each waiving of the tax is a reduction in the primal-tree distance to some node and its descendents, which corresponds in the dual tree to two path-label-altering operations. Each deletion and each waiving of tax can be carried out in amortized $O(\log n)$ time. The overall running time for EfficientBoundarySpanner is therefore $O(n \log n)$.

7. DISTANCES AMONG NODES THAT BELONG TO A TREE

Theorem 7.1. *Let G be a plane graph with nonnegative edge-weights $\ell(\cdot)$. Let T be a tree in G. For any $\epsilon > 0$, there is a subgraph H of G of weight $O(\epsilon^{-4}\ell(T))$ such that, for every pair of nodes x, y in T, $\mathrm{dist}_H(x,y) \le (1+\epsilon)\ \mathrm{dist}_G(x,y)$.*

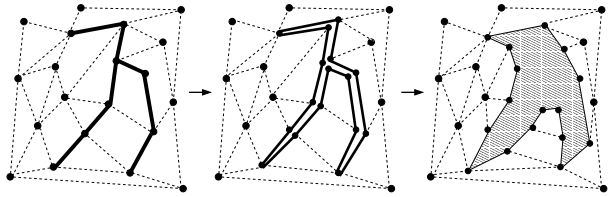

Figure 6: Cutting open a graph along a tree. (a) The solid edges represent a Steiner tree (b) Cut along the tree by duplicating each edge (and vertices as necessary) and separating the edges. (c) A new face is introduced: the shaded face. This face can now become the infinite face, and its boundary the boundary of the new graph.

PROOF. The *planar Euler tour* of T traverses each edge of T twice and visits each node v of T some number $n(v)$ of times. Modify G_0 by replacing each edge of T by two edges and replacing each node by $n(v)$ copies in such a way that the Euler tour becomes a simple cycle C. (See Figure 6.) The simple cycle C is the boundary of a face in G_1, and $\ell(C) = 2\ell(T)$. Transform the embedding so that this face is the infinite face. Let G_1 be the result. The theorem follows by applying Theorem 6.1 to G_1. □

Efficient construction

The Euler-tour construction can be carried out in linear time. The boundary-to-boundary spanner of Theorem 6.1 can be constructed in $O(\epsilon^{-1} n \log n)$. Thus the entire construction takes $O(\epsilon^{-1} n \log n)$ time.

8. PROOF OF MAIN THEOREM

Finally we prove our main theorem, Theorem 1.3. Given an n-node planar graph G, a Steiner tree T whose weight is at most twice the minimum can be found in $O(n \log n)$ time using the algorithm of Mehlhorn [18]. Applying the construction of Theorem 7.1 to G and T yields a subgraph G' with the properties specified in Theorem 1.3. The construction takes $O(\epsilon^{-1} n \log n)$ time.

Acknowlegements

Thanks to Glencora Borradaile and Claire Kenyon for helpful discussions.

9. REFERENCES

[1] U. A. Acar, G. E. Blelloch, R. Harper, J. L. Vittes, and S. L. M. Woo, " Dynamizing static algorithms, with applications to dynamic trees and history independence," *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 531–540, 2004

[2] S. Alstrup, J. Holm, K. D. Lichtenberg, and M. Thorup. "Maintaining information in fully-dynamic trees with top trees," *ACM Transactions on Algorithms* **1**(2), pp. 243-264, 2005.

[3] I. Althöfer, G. Das, D. Dobkin, D. Joseph, L. Soares, "On sparse spanners of weighted graphs," *Discrete and Computational Geometry*, **9**, pp. 81-100, 1993.

[4] S. Arora, "Polynomial-time approximation schemes for Euclidean TSP and other geometric problems," *Journal of the ACM* **45**, pp. 753-782, 1998.

[5] S. Arora, M. Grigni, D. R. Karger, P. N. Klein, A. Woloszyn, " A polynomial-time approximation scheme for weighted planar graph TSP," *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 33-41, 1998.

[6] R. F. Cohen, R. Tamassia, "Combine and Conquer," *Algorithmica* **18**(3), pp. 324-362 (1997).

[7] G. Das and D. Joseph, "Which triangulations approximate the complete graph?" *Proceedings of the International Symposium on Optimal Algorithms*, Springer LNCS 401, pp. 168-192, 1989.

[8] G. Das, G. Narasimhan, and J. S. Salowe, "A new way to weight malnourished Euclidean graphs," *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms* , pp. 215-222, 1995.

[9] E. D. Demaine, M. Hajiaghayi, "Bidimensionality: new connections between FPT algorithms and PTASs," *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 590–601, 2005.

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

[11] M. Grigni, E. Koutsoupias, and C. H. Papadimitriou, "An approximation scheme for planar graph TSP," *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pp 640–645, 1995.

[12] M. Grigni, "Approximate TSP in graphs with forbidden minors," *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming*, pp. 869-877, 2000.

[13] P. N. Klein, "Preprocessing an undirected planar network to enable fast approximate distance queries," *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Algorithms*, pages 820–827, 2002.

[14] P. N. Klein, "Multiple-source shortest paths in planar graphs," *Proceedings of the Sixteenth ACM-SIAM Symposium on Discrete Algorithms*, pp. 146–155, 2005.

[15] P. N. Klein, "A linear-time approximation scheme for planar weighted tsp," *EEE Symposium on Foundations of Computer Science*, pp. 647–656, 2005.

[16] C. Levcopoulos and A. Lingas, "There are planar graphs almost as good as the complete graphs and as short as minimum spanning trees," *Proceedings of the International Symposium on Optimal Algorithms*, Springer LNCS 401, pp. 9-13, 1989.

[17] J. Mitchell, "Guillotine subdivisions approximate polygonal subdivisions: Part II– a simple PTAS for geometric k-MST, TSP, and related problems," *SIAM Journal on Computing* **28**, pp. 298–1309, 1999.

[18] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs," *Information Processing Letters* 27(3), pp. 125-128, 1988.

[19] C. H. Papadimitriou and M. Yannakakis, "The traveling salesman problem with distances one and two," *Mathematics of Operations Research*, **18**(1), pp. 1–11, 1993.

[20] S. B. Rao and W. D. Smith, "Approximating geometrical graphs via 'spanners' and 'banyans'," *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 540-550, 1998.

[21] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *Journal of Computer and System Sciences*, 26(3), pp. 362–391, 1983.

[22] M. Thorup, "Compact oracles for reachability and approximate distances in planar digraphs," *Journal of the ACM* **51**(6), pp. 993-1024, 2004.

[23] R. E. Tarjan and R. F. Werneck, "Self-adjusting top trees," *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pp. 813–822, 2005.