A linear-time approximation scheme for TSP in planar graphs with edge-weights

Philip N. Klein
Brown University

Abstract

We give an algorithm requiring $O(c^{1/\epsilon^2}n)$ time to find an ϵ-optimal traveling salesman tour in the metric defined by a planar graph with nonnegative edge-lengths.

1 Introduction

The traveling salesman problem is often the first problem researchers use to test a new optimization technique. [21] In a metric, a *tour* is a cycle $(v_0\, v_2\, \ldots\, v_{n-1})$ of the points on the metric, and the weight of the tour is the sum $\sum_{i=0}^{n} \text{dist}(v_i, v_{(i+1) \bmod n})$, where dist$(u, v)$ is the distance between u and v. The goal is to find the minimum-weight tour. The problem is MAXSNP-hard [23, 24] in arbitrary metrics, and the best approximation ratio known, that proved by Christofides[10], is 1.5. For the metric of an unweighted planar graph (one in which every edge has weight one), Grigni, Koutsoupias, and Papadimitriou [16] gave an algorithm that requires $n^{O(1/\epsilon)}$ to find an ϵ-optimal tour. Arora, Grigni, Karger, Klein, and Woloszyn [5] subsequently gave an approximation scheme for the more general problem in which the planar graph's edges have arbitrary nonnegative weights. Their algorithm requires $n^{O(\epsilon^{-2})}$ time. Both algorithms are somewhat complicated, and involve a recursive decomposition using new planar-separator lemmata. The latter paper introduced the idea of using a *spanner* result to handle edge-weights.

In parallel development, Arora [3] and Mitchell [20] showed that a PTAS also exists for *Euclidean TSP* (i.e., the subcase in which the points lie in $\Re^2$ and distance is measured using the Euclidean metric). This PTAS finds an ϵ-optimal tour in $n^{O(1/\epsilon)}$ time. Arora [4, 2] improved the running time of his algorithm to $O(n \cdot (\log n)^{O(1/\epsilon)})$, using randomization. Finally, Rao and Smith [25] gave a PTAS for the two-dimensional Euclidean case that takes time $O(\epsilon^{-O(\epsilon)}n + n\log n)$.

In view of the fact that an ϵ-optimal tour can be found in the Euclidean case in time that is polynomial with a fixed degree, independent of ϵ, it seems natural to ask whether the same holds true for the planar case. In this paper, we answer this question. Our algorithm finds an ϵ-optimal tour in time that is $O(c^{1/\epsilon^2}n)$ where c is a constant.

1.1 Other related work

Grigni and Sissokho ([17], building on [18]) have given a quasipolynomial approximation scheme for weighted TSP in minor-excluded graphs. This paper proved a spanner result for minor-excluded graphs. Berger, Czumaj, Grigni, and Zhao ([7], building on [12]) give a PTAS for the problem of finding a minimum-weight 2-edge-connected spanning sub-multigraph of an edge-weighted planar graph (duplicate edges are allowed), and a quasipolynomial approximation scheme for finding a minimum-weight 2-edge-connected or biconnected spanning subgraph of an edge-weighted planar graph. This paper introduced a new spanner construction.

Baker [6] gives a method for obtaining PTASs for a variety of optimization problems in planar graphs, e.g. maximum-weight independent set and minimum-weight vertex cover. The resulting algorithms are linear time (for fixed ϵ). The key idea (interpreted in modern parlance) is to turn a problem in a planar graph to a problem in a graph with bounded treewidth. The technique can handle *weights* on nodes/edges but the problems it can address are quite local in nature.

Demaine and Hajiaghayi [13] describe a framework for PTASs that is based on the notion of *bidimensionality*. They derive approximation schemes for subclasses of minor-excluded graphs that involve turning the input graph into a low-treewidth graph. Their results apply to graphs that are not planar. Their framework can be viewed as a way to generalize Baker's approach so as to derive algorithms for nonlocal problems, such as feedback vertex set and connected dominating set. For planar graphs in particular, they derive EPTASs for several (unweighted) problems. In relation to their framework, our result is an example of how one can more thoroughly exploit planarity to derive a fast and simple EPTAS.

1.2 The framework

In this paper, we present a simple framework for deriving PTASs for planar graphs. A key step of the framework is a simple technique we call *thinning*, adapted from Baker's method. Given a planar graph G whose edge-weights sum to W and given a parameter k, the *thinning* procedure selects a weight-W/k subset S of G's edges such that $G - S = H - S$ for some planar graph H of radius[1] at most k.

The framework can give rise to two forms of algorithm, depending on whether thinning is performed in the planar dual or the primal. (To our knowledge, this paper represents the first use of thinning in the dual.) If the former, the algorithm has the following four steps:

Spanner step: Delete some edges of the input graph while approximately preserving the optimal value.[2]

[1] The *radius* of an undirected graph is the minimum height of any rooted spanning tree of the graph.
[2] This was the also the first step in [5] and subsequently in, e.g., [17] and one of the algorithms of [7].

Thinning step: Apply thinning to the planar dual, effectively contracting some edges, which should not increase the optimal value.

Dynamic-programming step: Use dynamic programming to find the optimal solution in the dual of the thinned graph.

Lifting step: Convert the optimal solution found in the previous step to a solution for the pre-thinned graph by incorporating some of the edges contracted during thinning.

This is the form of the TSP algorithm. For TSP, edge deletion can increase the optimal value but we ensure in the spanner step that the increase is small. Edge contraction (in the thinning step) can only decrease the optimal value. The dynamic-programming step operates on a graph whose dual has bounded radius. This form is appropriate for connectivity problems such as TSP and minimum-weight two-edge-connected-sub-multigraph.

In the alternative form, the spanner step *contracts* edges; the thinning step is applied to the primal and hence *deletes* edges; the dynamic-programming step operates on a graph of bounded radius. This form is appropriate for cut problems such as multiterminal cut.

We show that the thinning step produces a graph that has constant *branch-width*, and consequently low tree-width. The dynamic-programming step can consequently be performed in linear time. In Section 4 we review the definition of branch-width and its relation to tree-width. In Section 6, we give a dynamic-programming algorithm that exploits planarity to obtain a better bound than is achievable using branchwidth alone.

The spanner step requires an algorithm that, given a n-node planar graph G_0 with edge-weights and given a parameter ϵ, deletes [alternatively, contracts] edges so as to obtain a graph G such that

S1: OPT$(G) \leq (1+\epsilon)$OPT(G_0), and

S2: weight$(G) \leq \rho_\epsilon \cdot$ OPT(G_0)

where OPT(G) is the value of the optimum for input graph G, and weight(G) is the sum of weights of edges in G. Since the running time of the dynamic-programming step is at least exponential in ρ_ϵ, obtaining a polynomial running time depends on ρ_ϵ being $O(\log n)$ for fixed ϵ, and obtaining a linear running time depends on ρ_ϵ being $O(1)$. For TSP, $\rho_\epsilon = O(1)$, as we discuss in Section 3.

A trivial spanner result holds for problems in which weight$(G_0) \leq \rho_\epsilon$OPT(G_0), e.g. unweighted TSP. For such a problem, our approach can potentially yield algorithms that run in $O(c^{1/\epsilon} n)$ time.

The lifting step should increase the cost by at most

$$c \cdot \text{weight of edges eliminated during thinning} \qquad (1)$$

for some constant c. In the case of TSP, we show in Section 4 that this holds with $c = 2$.

Now we consider in more quantitative detail an algorithm in the framework. Given an n-node input graph G_0 and parameter ϵ, the spanner step eliminates edges to obtain a graph G with properties S1 and S2 above. The elimination must be such that a solution for G is a solution for G_0. The thinning step, applied with $k := c\epsilon^{-1} \cdot \rho(\epsilon, n)$, eliminates a set of edges having weight $(1/k) \cdot \text{weight}(G)$, obtaining a graph H whose dual has radius k and such that $\text{OPT}(H) \leq \text{OPT}(G)$. The dynamic-programming step solves the problem optimally in time that is exponential in the radius bound k and linear in n. The lifting step turns the optimal solution for H into a solution for G (and hence for G_0) while increasing the weight of the solution by (1), which by choice of k is at most $\epsilon \cdot \text{OPT}(G_0)$. Since $\text{OPT}(H) \leq \text{OPT}(G) \leq (1+\epsilon)\text{OPT}(G_0)$, it follows that the solution obtained for G has weight $\leq (1+\epsilon)\text{OPT}(G_0) + \epsilon \cdot \text{OPT}(G_0)$. This analysis is done more formally for TSP in Section 5.

2 Preliminaries

In this section, we describe the basic definitions and results on planar embeddings and planar duals. Most of the material is standard, but we also introduce a variant of contraction that we call *compression*, and state some related results. In Subsection 2.1, we give some definitions and results that help us reformulate the TSP.

The traditional geometric definition of planar embeddings involves drawings of a graph on the plane. Proofs and algorithms become simpler when one uses an alternative definition of embedded planar graphs, a combinatorial definition. See [22].

For any given finite set E, we can interpret E as a set of edges, and we define $E \times \{\pm 1\}$ to be the corresponding set of *darts*. For each edge e, the darts of e, namely $\langle e, 1\rangle$ and $\langle e, -1\rangle$, represent the two opposite orientations of e. We define $\text{rev}(\cdot)$ to be the function that takes each dart to the corresponding dart in the opposite direction: $\text{rev}(\langle e, i\rangle) = \langle e, -i\rangle$.

We define an embedded graph on E to be a pair $G = \langle \pi, E\rangle$ where π is a permutation of the darts of E. The permutation cycles of π are called the *nodes*[3] of G. Each node v is a permutation cycle $(d_1\ d_2\ \ldots\ d_k)$, and we use $D(v)$ to denote the set $\{d_1, d_2, \ldots, d_k\}$. We use $E(G)$ and $V(G)$ to denote, respectively, the set of edges and the set of nodes of a graph G.

For a dart d of G, we define $\text{tail}_G(d)$ to be the orbit of π containing d. We define $\text{head}_G(d) = \text{tail}_G(\text{rev}(d))$. For an edge e of G, we define $\text{ends}_G(e) = \{\text{head}_G(\langle e, 1\rangle), \text{tail}_G(\langle e, 1\rangle)\}$.

A *walk* of darts in G is a sequence $d_1 \ldots d_k$ of darts such that, for $i = 2, \ldots, k$, $\text{head}_G(d_{i-1}) = \text{tail}(d_i)$. The *start* of the walk is $\text{tail}_G(d_1)$ and the *end* is $\text{head}_G(d_k)$. It is a *closed* walk if in addition $\text{head}_G(d_k) = \text{tail}_G(d_1)$. It is a simple path/cycle if no node occurs twice as the head of a dart (cycle if closed, path if not). The walk, path, or cycle is said to contain an edge e if it contains a dart of e. It is said to contain a node v if v is the head or tail of some dart in the sequence. We define $\text{rev}(d_1 \ldots d_k) = \text{rev}(d_k) \ldots \text{rev}(d_1)$.

We denote by $E(P)$ and $V(P)$ respectively the set of edges contained by P and the set of nodes contained by P. A walk/path whose start is u and whose end is v is called a u*-to-*v walk/path.

[3]Note that nodes are defined in terms of edges, rather than the other way round. This definition precludes isolated nodes.

To define the faces of the embedded graph, we define another permutation π^* of the set of darts by composing π with rev: $\pi^* = \pi \circ \text{rev}$. Then the *faces* of the embedded graph $\langle \pi, E \rangle$ are defined to be the permutation cycles of π^*. Note that a face of G can be interpreted as a closed walk in G.

We say that an embedding π of a graph G is *planar* if it satisfies Euler's formula: $n - m + \phi = 2\kappa$, where n=number of nodes, m=number of arcs, ϕ=number of faces, and κ=number of connected components. In this case, we say $G = \langle \pi, E \rangle$ is a *planar embedded graph*.

The dual of a connected embedded graph $G = \langle \pi, E \rangle$ is defined to be the embedded graph $G^* = \langle \pi^*, E \rangle$. Since rev $\circ$ rev is the identity, we obtain the following.

Proposition 1 $G^{**} = G$.

It can be shown that the dual of a connected graph is connected. It follows that the connected components of G^* correspond one-to-one with the connected components of G. Hence if G satisfies Euler's formula then so does G^*. Thus the dual of a planar embedded graph is a planar embedded graph.[4]

Let T be a spanning tree of G. For an edge $e \notin T$ whose endpoints are in S, there is a unique simple cycle consisting of e and the unique path in T between the endpoints of e. This cycle is called the *elementary cycle* of e with respect to T in G.

For a spanning tree T of G, we denote by T^* the set of edges of G that are not in T. The following is a classical result.

Proposition 2 *If G is a planar embedded graph and T is a spanning tree of G, then T^* is a spanning tree of G^*.*

We refer to T^* as the tree dual to T.

If $S \subset V(G)$, we use $\Gamma_G(S)$ to denote the set of edges e such that in G the edge e has one endpoint in S and one endpoint not in S. A set of this form is called a *cut* of G. Note that $\Gamma_G(S) = \Gamma_G(V(G) - S)$.

If S is connected in G and $V(G) - S$ is connected in G, we call $\Gamma_G(S)$ a *bond*.

Proposition 3 *If G is a planar embedded graph, the edges of a bond in G form a simple cycle in G^* and vice versa.*

It follows from Proposition 3 that every simple cycle C in G defines a bipartition of the faces of G; namely the bipartition $(S, V(G) - S)$ where $E(C) = \Gamma_{G^*}(S)$.

Let f_∞ be a face of G. We call f_∞ the *infinite face* by analogy to geometric embeddings. For combinatorial embeddings, the choice is arbitrary. We say the simple cycle C *encloses* a face f with respect to f_∞ if f belongs to the set S such that $E(C) = \Gamma_G(S)$ and $f_\infty \notin S$. We say that C strictly encloses an edge with respect to f_∞ if the edge belongs to a face enclosed by C but does not belong to C.

[4]For disconnected graphs, this definition of dual diverges from the geometric definition in that it assigns multiple dual nodes to a single region of the sphere/plane. According to the geometric definition, the dual of a graph is always connected. However, choosing that definition means giving up, for example, the nice property that $G^{**} = G$.

Corollary 4 *Let G be a connected plane graph, let T be a rooted spanning tree of G, let v be a nonroot node of G, and let e be the parent edge of v. Then the elementary cycle of e in G^* with respect to T^* consists of the edges of $\Gamma_G($descendents of v in $T)$*

Proposition 5 *An edge e is a self-loop of G iff it is a cut-edge of G^*.*

We discuss two ways of removing edges from an embedded graph, deleting and compressing, both of which preserve planarity.

Deleting an edge e of an embedded graph $G = \langle \pi, E \rangle$ is an operation that produces the graph $G' = \langle \pi', E' \rangle$ where $E' = E - \{e\}$ and, for each dart of E',

$$\pi'[d] = \begin{cases} \pi[\pi[d]] & \text{if } \pi[d] \text{ is a dart of } e \\ \pi[d] & \text{otherwise} \end{cases}$$

For a set S of edges, we denote by $G - S$ the embedded graph obtained by deleting the edges of S. The order of deletion does not affect the final embedded graph. It is easy to see that deletion preserves planarity.

We define edge *compression* to be deletion in the dual. That is, compressing an edge e of G is an operation that produces the graph $(G^* - \{e\})^*$. We denote the result as $G/\{e\}$. Since deletion preserves planarity and the dual of a plane graph is a plane graph, compression preserves planarity. The operations of deletion and compression commute.

Figure 1 illustrates the effect of edge compression on the underlying graph in three examples. If e is not a self-loop in G then the effect of compressing e in G is to contract e as shown in the top left diagram. The thick line represents the edge to compress. If e is a self-loop in G, so a cut-edge in G^*, and is not the only edge incident to either of its endpoints then the effect to duplicate v, as shown in the bottom left diagram; one copy has as its incident edges those edges that in G are incident to v and strictly enclosed by e (with respect to some designated face f_∞) and the other copy has as its incident edges those edges that in G are incident to v and not enclosed by e (and not equal to e). If e is a self-loop in G and is the only edge incident to one of its endpoints in G^*, the effect is to delete e.

2.1 Preliminaries related to TSP

If G is a connected graph with edge-weights, a *tour* for G is a closed walk that contains all nodes of G. If G is a graph that is not necessarily connected, we define a *multitour* of G to be a sequence W of darts such that, for each connected component of G, the (not necessarily consecutive) subsequence of darts of W belonging to that component is a tour of that component. For a multitour W and a set S of edges, define $W - S$ to be the subsequence of W in which elements of S are omitted. For an assignment weight$(\cdot)$ of nonnegative weights to the edges of G and a set S of edges, define weight$(S) = \sum\{\text{weight}(e) \ : \ e \in S\}$. For a subgraph H, define weight$(H) = $ weight$(E(H))$. Define

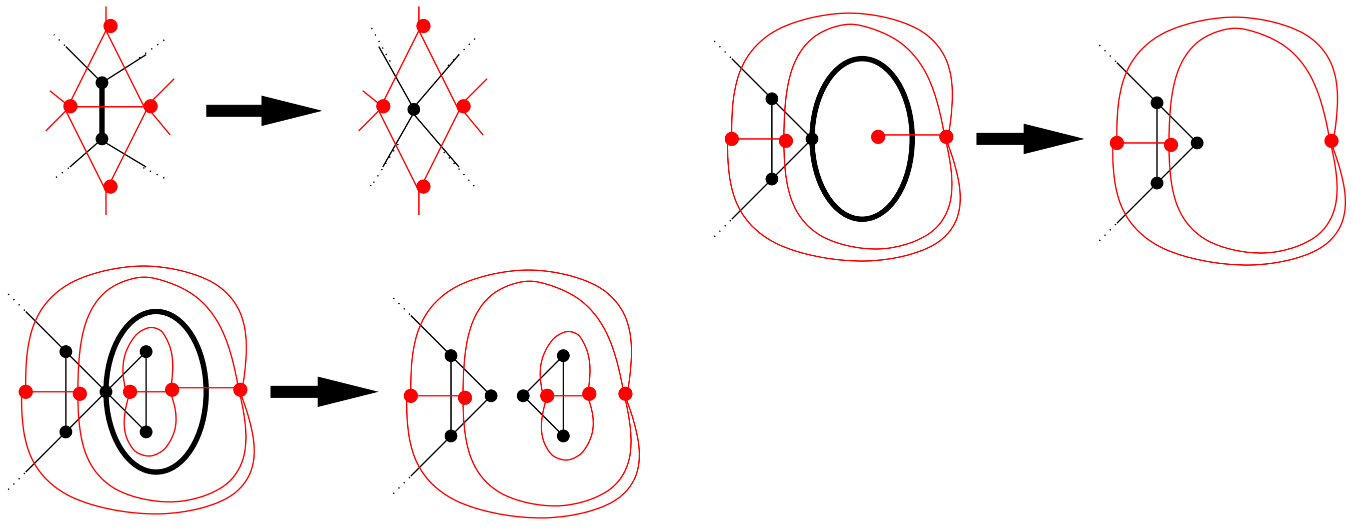


Figure 1. Three examples of compression.

OPT$(G, \text{weight}(\cdot))$ to be the minimum weight of a multitour of G. We omit the second argument when doing so creates no ambiguity.

Lemma 6 *Let G be an embedded graph, let S be a set of edges of G, and let W be a multitour of G. Then $W - S$ is a multitour of G/S.*

Proof: By induction, it suffices to consider a singleton set $S = \{e\}$. Suppose e is not a self-loop in G. In this case, $G/\{e\}$ is the graph obtained by contracting e. Since the endpoints of e in G are coalesced in $G/\{e\}$, if e belongs to W then removing e from W preserves the multitour property. If e does not belong to W, then clearly W is a multitour of $G/\{e\}$.

Suppose e is a self-loop in G with endpoint v. Then e is a cut-edge in G^*. If in G^* the edge e is the only edge incident to one of its endpoints then $G/\{e\} = G - \{e\}$. In this case, if e belongs to W then removing e from W preserves the mulitour property; if e does not belong to W, then W is a multitour of $G/\{e\}$.

If in G^* the edge e is not the only edge incident to either endpoint, then $G/\{e\}$ is the graph obtained from $G - \{e\}$ by splitting v into two copies, v_1 and v_2, and reattaching each edge incident to v in G to either v_1 or v_2. An edge strictly enclosed by e in G is attached to v_1 and an edge not enclosed by e is attached to v_2. This splits the component of G containing v into two components K_1 and K_2. Observe that in G every path between K_1 and K_2 must pass through v.

Let W' be the subsequence of $W - \{e\}$ consisting of darts belonging to the component of G containing e. Write $W' = A_0 \circ A_1 \circ \cdots \circ A_{k-1}$ where each A_i is a nonempty maximal consecutive subsequence of darts of one of the two components K_1, K_2 of $G/\{e\}$. By the observation, since W' is a tour, the end node of each A_i is v. It follows that W' has the multitour property for the graph consisting of K_1 and K_2, so W is a multitour for $G/\{e\}$. $\square$

Lemma 6 shows that OPT$(G/S) \leq$ OPT(G).

For a walk W of darts, define the multiplicity of an edge e in W, denoted multi(e, W), to be the number of occurences of darts of e in W.

Lemma 7 *Let W be a tour in G, and suppose some dart d occurs at least twice in W. Then there exists a tour $\hat{W}$ in G such that, for each edge e,*

$$\textit{multi}(e, \hat{W}) = \begin{cases} \textit{multi}(e, W) - 2 & \textit{if } d \textit{ is a dart of } e \\ \textit{multi}(e, W) & \textit{otherwise} \end{cases}$$

Proof: Write $W = W_1 \circ d \circ W_2 \circ d$. Then $W_1 \circ$ rev(W_2) is a tour of G. □

Lemma 7 shows that, in seeking the minimum-weight walk containing a given set of nodes, we can restrict ourselves to considering walks using each dart at most once.

We define an *Eulerian multisubgraph* of a graph G to be a multiset $\mathcal{E}$ of edges of G such that

- the edges of $\mathcal{E}$ within each connected component of G are themselves connected, and
- for every node v of G, the number of edges of $\mathcal{E}$ incident to v is even.

The weight of an Eulerian multisubgraph is the sum of the weights of the edges, counting multiplicities. We call it a *bisubgraph* if the maximum multiplicity is two.

Note that for a closed walk W in a graph G, for each node v, $|\{d \in W \ : \ \text{tail}_G(d) = v\}| = |\{d \in W \ : \ \text{head}_G(d) = v\}|$. This observation together with Lemma 7 implies that for the minimum-weight walk W there is an Eulerian bisubgraph with the same multiplicities. Conversely,

Proposition 8 (Euler) *There is a linear-time algorithm that, given a graph G and an Eulerian bisubgraph $\mathcal{E}$ of G, outputs a walk in G with the same multiplicities.*

Proposition 8 shows that the problem of finding a minimum-weight walk containing a given set of nodes is equivalent to the problem of finding a minimum-weight Eulerian bisubgraph containing a given set of nodes.

Proposition 9 *For any edge-weighted planar embedded graph there exists an minimum-weight multi-tour that uses each dart at most once and does not cross itself.*

3 Spanner

Althöffer, Das, Dobkin, Joseph, and Soares [1] considered a simple and general procedure for producing a spanner in a (not necessarily planar) graph G_0: start with an empty graph G, consider the edges of G_0 in increasing order of weight, and add an edge to G if the edge's weight was much smaller than the minimum-weight path in H between its endpoints. They did not address the exact running time of the procedure, but it clearly consists of $O(n)$ iterations, each involving a shortest-path computation.

For planar graphs, therefore, it runs in $O(n^2)$ time [19]. They proved several results about the size and weight of the resulting spanner, including the following result that is specific to planar graphs.

Theorem 1 (Althöffer et al.) *For any planar graph G_0 with edge-weights and any $\epsilon > 0$, there is an edge subgraph G such that*

S1: *weight$(G) \leq (1 + 2\epsilon^{-1})MST(G_0)$, where $MST(G_0)$ is the weight of the minimum spanning tree of G_0, and*

S2: *for every pair of nodes u and v,*

$$\begin{aligned} &\textit{minimum weight of a } u\textit{-to-}v \textit{ path in } G \\ \leq\ & (1+\epsilon) \cdot \textit{minimum weight of a } u\textit{-to-}v \textit{ path in } G_0 \end{aligned} \tag{2}$$

By exploiting planarity, we can give an algorithm that runs in linear time but that can be shown (using the same analysis technique used by Althöffer et al.) to achieve the same properties.

```
define SPANNER($G_0, \epsilon$):
    let $x[\cdot]$ be an array of numbers, indexed by edges
    find a minimum spanning tree $T$ of $G_0$
    assign $x[e]$ := weight($e$) for each edge $e$ of $T$
    let $T^*$ be the dual tree, rooted at the infinite face
    for each edge $e$ of $T^*$, in order from leaves to root
        let $f_e$ be the face of $G_0$ whose parent edge in $T^*$ is $e$
        let $e$=$e_0, e_1, \ldots, e_s$ be the sequence of edges comprising $f_e$
        $x_{\text{omit}} := \sum_{i=1}^{s} x[e_i]$
        $x[e]$ := if $x_{\text{omit}} \leq (1+\epsilon)$weight($e$) then $x_{\text{omit}}$
                  else weight($e$)
    return the set of edges $e$ such that $x[e]$ = weight($e$)
```

The minimum spanning tree of G_0 can be found in linear time using the algorithm of Cheriton and Tarjan [9].

Now we address correctness of the procedure. Say an edge e is *accepted* when $x[e]$ is assigned weight(e), and *rejected* otherwise.

Lemma 10 *In the for-loop iteration in which e is considered, for every other edge e_i of f, $x[e_i]$ has been assigned a number.*

Proof: The face f_e has only one parent edge in T^*, and it is e. For every other edge e_i of f_e, either e_i belongs to T or e_i is a child edge of f_e in T^*. □

For any edge e of G_0 not in T,

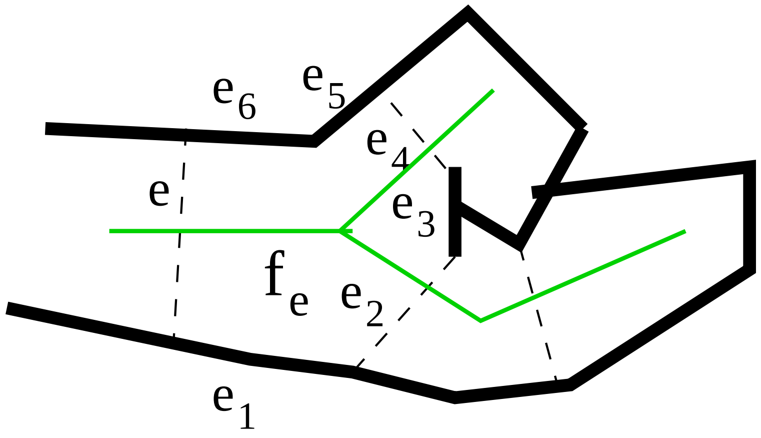


Figure 2. Diagram showing part of dual tree (in light edges) and primal tree (in dark edges) and primal nontree edges (dashed): e_2 and e_4 are child edges of e in the dual tree. The face f_e is indicated.

- let $\hat{G}_e$ denote the subgraph of G_0 consisting of accepted edges together with e,
- let $\hat{f}_e$ denote the face of $\hat{G}_e$ that contains e and encloses f_e,
- let $\hat{W}_e$ denote the walk formed by the sequence of edges comprising $\hat{f}_e$ not including e itself, and
- let $P_e = \begin{cases} e & \text{if } e \text{ is accepted} \\ \hat{W}_e & \text{otherwise} \end{cases}$

Note that each of $\hat{W}_e$ and P_e has the same endpoints as e. For an edge e of T, define $P_e = e$.

Lemma 11 *For any edge e of G_0, not in T,*

1. *every edge of $\hat{f}_e$ is either in T or is a descendent of e in T^*, and*
2. *$\hat{W}_e = P_{e_1} \circ \cdots \circ P_{e_s}$, where $e_1 \ \ldots \ e_s$ is the walk consisting of the edges comprising f_e other than e.*

Proof: by induction. Consider the case in which e is a leaf-edge of T^*. Let f be the corresponding leaf-node in G_0^*. Because f is a leaf, the only incident edge that is in T^* is e itself, so $e_1, \ldots, e_s$ belong to T. All these edges are accepted, proving Part 1. To prove Part 2, note that $W_e = e_1 \ \cdots \ e_s$ and that $P_{e_i} = e_i$ for $i = 1, \ldots, s$. Thus the lemma holds for e.

Consider the case where e is not a leaf. Let $\hat{G}_{e+}$ be the subgraph of G_0 consisting of accepted edges together with $e, e_1, \ldots, e_s$. For each e_i, recall that $\hat{f}_{e_i}$ is the face of $\hat{G}_{e_i}$ that contains e_i and encloses f_{e_i}. We claim that $\hat{f}_{e_i}$ is also a face of $\hat{G}_{e+}$. To prove the claim, note that $\hat{G}_{e_i}$ can be obtained from $\hat{G}_{e+}$ by deleting a subset of $\{e, e_1, \ldots, e_s\} - \{e_i\}$. None of these edges are edges of T or descendents in T^* of e_i, so, by Part 1 of the inductive hypothesis, none belongs to $\hat{f}_{e_i}$.

Note that $\hat{G}_e$ can be obtained from $\hat{G}_{e+}$ by deleting those edges among $e_1, \ldots, e_s$ that are rejected. By the claim, each such deletion replaces a rejected edge e_i in f_e with the walk $\hat{W}_{e_i}$. This together with the definition of P_{e_i} proves Part 2. By Part 1 of the inductive hypothesis, every edge in each $\hat{W}_{e_i}$ is an edge of T or a descendent of e_i in T and hence a descendent of e as well. This proves Part 1. $\square$

Lemma 12 *In the for-loop iteration that considers e,*

- *the value assigned to x_{omit} is weight$(\hat{W}_e)$, and*
- *the value assigned to $x[e]$ is weight(P_e).*

Proof: The proof is by induction. By Lemma 10, the edges $e_1, \ldots, e_s$ are considered before e. By the inductive hypothesis, $x[e_i] = $ weight(P_e). By Lemma 11, weight$(\hat{W}_e) = \sum_{i=1}^{s} x[e]$, which proves the first statement. The second statement follows by definition of P_e. $\square$

Corollary 13 *For each edge e, weight$(P_e) \leq (1+\epsilon)$weight(e).*

Proof: If e is accepted, $P_e = e$ so the statement holds trivially. Suppose e is rejected. By the conditional in the algorithm, in the iteration considering e, the value assigned to x_{omit} was at most $(1+\epsilon)$weight(e). By the first part of Lemma 12, weight$(\hat{W}_e)$ and therefore weight(P_e) are at most $(1+\epsilon)$weight(e). $\square$

Corollary 14 *The graph of accepted edges satisfies Property S2.*

Proof: For any pair of nodes u and v, let P be the shortest u-to-v path in G_0. For each edge e of P,there is a walk P_e consisting of accepted edges between the endpoints of e. By Corollary 13, weight$(P_e) \leq (1+\epsilon)$weight(e). Replacing each edge e of P with P_e therefore yields a walk of weight at most $\sum_{e \in P}(1+\epsilon)$weight$(e)$, which is at most $(1+\epsilon)$weight(P). $\square$

Lemma 15 *At any time during the algorithm's execution, the weight of the infinite face in the graph consisting of accepted edges is at most*

$$2 \cdot MST(G_0) - \epsilon \cdot \textit{weight}(\textit{accepted edges not in } T)$$

Proof: The proof is by induction. Before the for-loop commences, the graph of accepted edges is T, the minimum spanning tree of G_0. Hence the weight of the infinite face is exactly $2 \cdot MST(G_0)$, so the lemma's statement holds for this time. Consider a for-loop iteration, and let e be the edge being considered. If e is not accepted, there is no change to the set of accepted edges, so the lemma's statement continues to hold.

Suppose e is accepted. Let G_{after} be the subgraph consisting of edges accepted so far, and let $G_{\text{before}} = G_{\text{after}} - \{e\}$. Note that G_{after} can be obtained from G_e by deleting edges that will be accepted in the future. By the leaves-to-root ordering, none of the deleted edges are descendents of e in T^*. By Part 1 of Lemma 11, therefore, $\hat{f}_e$ is a face of G_{after}. Let g be the other face of G_{after} that contains e.

We claim that g is the infinite face of G_{after}. To prove the claim, note that G_{after} can be obtained from G_0 by deleting edges that have already been rejected and edges not yet considered. By the leaves-to-root ordering, e's proper ancestors in T^* have not yet been considered, so they are among the edges deleted. These deletions are contractions in the dual. The root of T^* is the infinite face, so the contractions result in g being the infinite face.

Note that G_{before} can be obtained from G_{after} by deleting e. This deletion replaces e in the face g with $\hat{W}_e$. This shows that

$$\begin{aligned}
&\text{weight of infinite face in } G_{\text{before}} - \text{weight of infinite face in } G_{\text{after}} \\
&\quad = \text{weight}(\hat{W}_e) - \text{weight}(e) \\
&\quad > (1+\epsilon)\text{weight}(e) - \text{weight}(e) \text{ because } e \text{ was accepted} \\
&\quad = \epsilon \cdot \text{weight}(e)
\end{aligned}$$

which shows that the lemma's statement continues to hold. □

Corollary 16 *The graph G of accepted edges satisfies Property S1.*

Proof: By Lemma 15, the weight of the infinite face in the graph consisting of all accepted edges is at most

$$2 \cdot MST(G_0) - \epsilon \cdot \text{weight}(\text{accepted edges not in } T)$$

so weight(accepted edges not in T) $\leq 2\epsilon^{-1} \cdot MST(G_0)$. Since weight$(T) = MST(G_0)$, it follows that the weight of all accepted edges is at most $(1 + 2\epsilon^{-1})MST(G_0)$. □

4 Thinning

In this section we describe how to remove a low-weight set of edges from a planar graph (by deletion or contraction) so as to obtain a graph in which it is simple to solve optimization problems.

The following lemma is implicit in Baker's approach [6].

Lemma 17 (Thinning Algorithm) *There is a linear-time algorithm that, for any planar embedded graph G, edge-weight assignment weight$(\cdot)$, and integer k, returns an edge-set S of weight at most $(1/k)$weight(G) and a planar embedded graph H with a spanning tree of height at most k such that $H - S = G - S$ and $|V(H)| = |V(G)|$ and $|E(H)| = |E(G)|$.*

Proof: Assume without loss of generality that G is connected. The algorithm is as follows. Carry out breadth-first search on G from some node r, obtaining a breadth-first search tree T and a labeling of the nodes with their distances from r in G. (Breadth-first search interprets each edge as having length 1.) Define the *level* of an edge e to be i if one endpoint has distance i from r and the other endpoint has distance $i-1$.

For $i = 0, 1, \ldots, k-1$, let S_i denote the set of edges e whose levels are congruent to i mod k. Let $t = \text{minarg}_i \text{weight}(S_i)$.

For $j = 0, 1, 2, \ldots$, let $\hat{H}_j$ be the graph obtained from G by deleting all nodes at distances greater than $(j+1)k+t$, and contracting every edge e of the breadth-first search tree T whose level is less than $jk+t$. The contractions coalesce into a single root all nodes at distances less than $jk+t$. (In the case in which $jk+t = 0$, there are no nodes at distance less than $jk+t$, but in this case $\hat{H}_j$ is rooted at r, and we take r to be the root.) Combine the planar embedded graphs $\hat{H}_1, \hat{H}_2, \ldots$ to form the planar embedded graph H by identifying the roots to form a single node $\hat{r}$.

Now we prove the correctness of the algorithm. Every edge is in at most one of the sets $S_0, \ldots, S_{k-1}$, so $\sum_{i=1}^{k} \text{weight}(S_i) \leq \text{weight}(G)$. Hence $\text{weight}(S_t) \leq (1/k)\text{weight}(G)$.

Now we show H has a spanning tree T' of height k. First, for $j = 0, 1, \ldots$, define T_j to be the spanning tree of $\hat{H}_j$ consisting of all edges of T remaining in $\hat{H}_j$. Every node of $\hat{H}_j$ except the root had distance from the root at most $(j+1)k+t$ in T so has distance at most k from the supernode in T_j. By merging the trees $T_1, T_2, \ldots$ at the roots, we get a tree that has height k and spans all the nodes in H.

For $j = 0, 1, \ldots$, let H_j denote the subgraph of G induced by the set of nodes at distance in $[jk+t, (j+1)k+t]$. The connected components of $G - S$ are the subgraphs $H_1, H_2, \ldots$. Each graph H_j can also be obtained from the corresponding graph $\hat{H}_j$ by deleting edges of S. This shows $G - S = H - S$.

□

A *tree decomposition* of a graph G is a pair (T, ϕ) where T is a tree and ϕ is a function from $V(T)$ to subsets of $V(G)$, such that the following conditions are satisfied.

- $V(G) = \bigcup_{x \in V(T)} \phi(x)$,
- for every edge e of G, there is a node $x \in V(T)$ such that $\phi(x)$ contains both endpoints of e, and
- for every node $v \in V(G)$, the set of nodes $\{x \,:\, v \in \phi(x)\}$ is connected in T.

The *width* of (T, ϕ) is $\max_{x \in V(T)} |\phi(x)| - 1$.

The following result is implicit in Baker's work, [6] and explicit in [26]. See also [15].

Lemma 18 *A planar graph of radius k has treewidth at most $3k$.*

The following simple corollary, used by Baker [6], is not used in obtaining the TSP result but it could be useful in applying the framework to cut problems such as multiterminal cut.

Corollary 19 (Edge Deletion) *There is a linear-time algorithm that, for any planar embedded graph G, edge-weight assignment weight$(\cdot)$, and integer k, returns an edge-set S of weight at most $(1/k)$weight(G), a planar embedded graph H such that $H - S = G - S$, and a tree decomposition of H having width at most $3k$.*

We next review the definition of branch-width given by Seymour and Thomas [29]. For a graph G and a set X of edges, $\partial(X)$ denotes the set of nodes v of G such that at least one edge incident to v is in X and at least one is not.

For a finite set $\mathcal{X}$, a *carving* of $\mathcal{X}$ is a family $\mathcal{C}$ of subsets of $\mathcal{X}$ such that

- $\emptyset, \mathcal{X} \notin \mathcal{C}$,
- no two members of $\mathcal{C}$ cross, and
- $\mathcal{C}$ is maximal subject to 1 and 2.

Let G be a graph. The width of a carving $\mathcal{C}$ of $E(G)$ is $\max_{X \in \mathcal{C}} |\partial(X)|$. The *branch-width* of G is the minimum, over all carvings $\mathcal{C}$ of $E(G)$, of the width of $\mathcal{C}$.

In Section 6, for the sake of simplicity of presentation, we make direct use of the fact that thinning produces a planar embedded graph of bounded radius. However, for the purpose of supporting future use of the framework, we give a lemma bounding the branch-width of planar embedded graphs with bounded radius, and a corollary combining this lemma with Lemma 17.

Lemma 20 *Let G be a planar graph and let T be a spanning tree of G such that every simple path in T has length at most ℓ. Then the branch-width of the dual G^* is at most $\ell + 2$.*

A bound of $\lceil (3/2)(\ell+2) \rceil$ on the tree-width of G^* follows because Robertson and Seymour [27] show that the tree-width of a graph is at most $\lceil \frac{3}{2}\beta \rceil - 1$, where β is the branch-width of the graph. The proof of Lemma 20 will be given in Section 8.

Corollary 21 (Edge Compression) *There is a linear-time algorithm that, for any planar embedded graph G, edge-weight assignment weight$(\cdot)$, and integer k, returns an edge-set S of weight at most $(1/k)$weight(G) and a planar embedded graph H of branch-width at most $2(k+1)$ such that $H/S = G/S$, $|V(H)| = O(|V(G)| + |E(G)|)$ and $|E(H)| = |E(G)|$.*

Proof: Apply Lemma 17 to G^* to find a set S of edges of weight at most $(1/k)$weight(G) and a planar graph H^* such that $H^* - S = G^* - S$ and H^* has a spanning tree T^* of height at most k. By duality and definition of compression, $H/S = G/S$. Every simple path in the spanning tree has at most $2k$ edges, so by Lemma 20, H has branch-width at most $2k + 2$. $\square$

The proof of Lemma 20 gives an algorithm to construct the corresponding carving.

5 TSP algorithm

Now we describe the TSP algorithm.

Let G_0 be the input planar embedded graph, and let weight$(\cdot)$ be the input edge-weight assignment.

Step 1 (Spanner Step): Let ϵ_0 be the desired accuracy. Define $\epsilon = \epsilon_0/2$. Apply the algorithm of Section 3 to G_0 and weight$(\cdot)$ with parameter ϵ to get an edge subgraph G.

Step 2 (Thinning step): Apply the algorithm of Corollary 21 to G and weight$(\cdot)$ with $k = 2\epsilon^{-1}(1 + 2\epsilon^{-1})$ to get a set S of edges of weight at most $(1/k)$weight(G), a planar embedded graph H such that $H/S = G/S$, and a spanning tree of H having height at most k.

Step 3 (Dynamic programming): Let weight$_S(\cdot)$ be the edge-weight assignment obtained from weight$(\cdot)$ by changing the weights of edges of S to zero. Find a tour T of H that is optimal with respect to weight$_S(\cdot)$.

Step 4 (Lifting): Interpreting $T - S$ as an Eulerian bisubgraph of G/S, use the procedure of Section 7 to obtain an Eulerian bisubgraph of G.

Step 5: Use Proposition 8 to turn the Eulerian bisubgraph of G into a tour of G.

5.1 Running time

Assume the input graph G_0 has n nodes and no parallel edges, so it has $O(n)$ edges. Each step of the TSP algorithm except Step 3 requires $O(n)$ time. Because the graph H has branch-width at most $2k+2$, the optimal tour of H can be computed using dynamic programming in $O(c^{k \log k} n)$ time for a constant c. Cook and Seymour [11] use such a dynamic-programming algorithm to solve TSP in graphs of bounded branch-width. In Section 6, we give a more direct algorithm that exploits planarity to achieve $O(c^k n)$ time for a constant c. The choice of k yields $O(d^{1/\epsilon^2} n)$ time for a constant d.

5.2 Correctness

Lemma 22 *Let H be an edge-weighted graph and let S be a set of edges each having weight zero. Then OPT$(H) = OPT(H/S)$.*

Proof: By Lemma 6, OPT$(H/S) \leq$ OPT(H). Conversely, by Lemma 7 the optimum tour of H/S defines an Eulerian bisubgraph B of H/S that by Theorem 4 can be lifted to obtain a Eulerian bisubgraph B' of H such that $B = B' - S$. Since the edges of S have zero weight, weight(B') = weight(B). By Proposition 8, B' defines a tour of H having the same weight. □

Theorem 2 *The algorithm finds a tour of weight at most $(1 + \bar{\epsilon})OPT(G)$.*

Proof: First we show OPT$(G) \leq (1 + \epsilon)$OPT(G_0). (This argument was used in [5].) Let T_0 be an optimal tour of G_0. For each edge uv of T_0 that is not in G, there is a u-to-v path in G of weight at most $(1 + \epsilon)$weight(uv); replace uv in T_0 with that path. The result of all the replacements is a tour T_1 whose weight is at most $1 + \epsilon$ times that of T_0.

By correctness of the algorithm of Section 6 (Theorem 3),

$$\begin{aligned} \text{weight}(T) &= \text{OPT}(H') \\ &= \text{OPT}(H/S) && \text{by Lemma 22} \\ &= \text{OPT}(G/S) && \text{because } H/S = G/S \\ &\leq \text{OPT}(G) && \text{by Lemma 6} \\ &\leq (1+\epsilon)\text{OPT}(G_0) \end{aligned}$$

The lifting procedure increases the weight of the tour by at most

$$\begin{aligned} &2 \cdot \text{weight}(S) \\ &\leq (2/k) \cdot \text{weight}(G) && \text{by Lemma 17} \\ &\leq \epsilon(1 + 2\epsilon^{-1})^{-1}\text{weight}(G) && \text{by choice of } k \\ &\leq \epsilon \cdot MST(G_0) && \text{by Property S1 of Theorem 1} \\ &\leq \epsilon \cdot \text{OPT}(G_0) \end{aligned}$$

since a tour contains a spaning tree. Thus the Eulerian bisubgraph resulting from lifting has weight at most

$$(1 + \epsilon)\text{OPT}(G_0) + \epsilon \cdot \text{OPT}(G_0).$$

□

6 Solving TSP in a planar graph with bounded dual radius

In this section we describe an algorithm that, given an edge-weighted planar embedded graph H such that H^* has radius k, and given a set R of nodes, finds an minimum weight walk W such that $R \subset V(W)$. To find an optimal multitour, R is set to $V(H)$. Rather than describe the algorithm for this special case, we describe the algorithm for the more general case because doing so requires very little change.

Theorem 3 *There is an algorithm that, given a planar embedded graph H , an edge-weight assignment for H, a subset R of nodes of H, and a spanning tree T^* of H^* in which every simple path has length at most ℓ, finds an minimum-weight walk in H that visits all nodes in R. The algorithm takes time $O(c^\ell(|V(H)| + |E(H)|))$ for some constant c.*

6.1 Reduction to degree three

First we show how to reduce the problem to the case in which the degree of the input graph is bounded by three. Then we show how to solve this case using dynamic programming.

Step 1: Triangulate the faces of H^* by adding zero-weight artificial edges until every face has size at most three. Let A be the set of artificial edges added. Let $\hat{H}^*$ be the resulting planar embedded graph.

Step 2: H can be obtained from $\hat{H}$ by contracting the artificial edges, which merges some nodes. Let $\hat{R} = \bigcup_{v \in R}\{$nodes of $\hat{H}$ merged to form $v\}$.

Step 3: Let W be a minimum-weight walk of $\hat{H}$ that visits all nodes of $\hat{R}$.

Step 4: Return $W - A$.

Lemma 23 *$W - A$ is a tour such that weight$(W - A) =$ OPT(H).*

Proof: We have $H^* = \hat{H}^* - A$, so $H = \hat{H}/A$. By Lemma 6, $W - A$ is a tour of $\hat{H}/A$. Furthermore,

$$\begin{aligned} &\text{weight}(W - A) \\ &= \text{weight}(W) && \text{since edges of } A \text{ have weight 0} \\ &= \text{OPT}(\hat{H}) && \text{since } W \text{ is an optimal tour of } \hat{H} \\ &= \text{OPT}(\hat{H}/A) && \text{by Lemma 22} \\ &= \text{OPT}(H) && \text{because } H = \hat{H}/A \end{aligned}$$

□

6.2 Overview of dynamic program

Now we describe how to find an optimal tour of $\hat{H}$ visiting all nodes of $\hat{R}$. The graph H^* has a rooted spanning tree T^* in which every simple path has at most ℓ edges, and $\hat{H}^*$ is obtained from H^* by adding edges, so T^* is also a spanning tree of $\hat{H}$. Because every face of $\hat{H}^*$ is a triangle, $\hat{H}$ has degree at most three. Let $\hat{T}$ be the set of edges of $\hat{H}$ not in T^*. Then $\hat{T}$ is a spanning tree of $\hat{H}$ and hence has degree at most three. Root $\hat{T}$ at a node r of degree 1 in T. The dynamic program will work up $\hat{T}$ from the leaves to the root. For each edge of $\hat{T}$, the dynamic program will construct a table. The value of OPT$(\hat{H})$ will be be computed from the table associated with the edge connecting the root to its child. Once the value of OPT$(\hat{H})$ is known, the tour itself can be constructed in a post-processing phase by working down from the root to the leaves. (The post-processing is straightforward, and we do not describe it here.)

6.3 Terminology

Before giving a detailed description of the tables, we need to introduce some terminology.

Traversals Let $\Gamma_{\hat{H}}(S)$ be a cut. We say a nonempty, dart-disjoint set $\mathcal{P}$ of walks in $\hat{H}$ is a *traversal of S in $\hat{H}$* if

- the start node and end node of each path are not in S,
- the internal nodes of each path are in S.

It follows that the first and last edges of each path belong to $\Gamma_{\hat{H}}(S)$. Define

$$\kappa(\mathcal{P}) = \{\{\text{first edge of } P, \text{ last edge of } P\} \ : \ P \in \mathcal{P}\}$$

Define

$$\begin{aligned}\text{weight}(\mathcal{P}) &= \sum\{\text{weight}(d) \ : d \in \mathcal{P}, d \text{ not a dart of } \Gamma_{\hat{H}}(S)\} \\ &\quad + \frac{1}{2}\sum\{\text{weight}(d) \ : d \in \mathcal{P}, d \text{ a dart of } \Gamma_{\hat{H}}(S)\}\end{aligned}$$

Configurations A *configuration* K of a cut $\Gamma_{\hat{H}}(S)$ is a nonempty multiset of unordered pairs $\{e_i, e_j\}$ such that each edge of $\Gamma_{\hat{H}}(S)$ occurs at most twice. The number of configurations is at most $m!$, where $m = |\Gamma_{\hat{H}}(S)|$.

If S is connected in $\hat{H}$ then the embedding determines a cyclic ordering of the edges of $\Gamma_{\hat{H}}(S)$, say $(e_1 \ \cdots \ e_m)$. In this case, we say that a configuration is *crossing* if it includes pairs (e_p, e_q) and (e_r, e_s) such that $p < r < q < s$. A Catalan bound shows that the number of noncrossing configurations is $2^{O(m)}$. This is where planarity is used in the dynamic program.[5]

For a configuration K, define weight(K) to be the sum of the weights of the edges in K, counting multiplicities.

6.4 Definition of the tables

In this subsection we describe the tables produced by the dynamic program. For each edge e of $\hat{T}$, let v_e denote the child endpoint of e, and let D_e denote the descendents of v_e. By Corollary 4, the edges comprising $\Gamma_{\hat{H}}(D_e)$ are exactly the edges comprising the elementary cycle of e in $\hat{H}^*$ with respect to T^*. (See Figure 3.) That elementary cycle consists of e together with a simple path in T^* between the endpoints of e. The cycle therefore contains at most $\ell + 1$ edges. This shows $|\Gamma_{\hat{H}}(D_e)| \leq \ell + 1$.

For a cut $\Gamma_{\hat{H}}(S)$ of $\hat{H}$ where S is connected in $\hat{H}$, for a configuration K of $\Gamma_{\hat{H}}(S)$, define

$$M_S(K) = \min\{\text{weight}(\mathcal{P}) \ : \quad \begin{array}{l}\kappa(\mathcal{P}) = K, \\ \mathcal{P} \text{ is a traversal of } S, \text{ and} \\ S \cap \hat{R} \subset V(\mathcal{P})\}\end{array}$$

We show in Corollary 27 that, for each edge e of T, the dynamic program will construct a table TAB_e, indexed by the noncrossing configurations K of $\Gamma_{\hat{H}}(D_e)$, such that $\text{TAB}_e[K] = M_{D_e}(K)$.

[5]Noncrossing configurations and a Catalan bound were used in a dynamic program for TSP by Arora et. al [5]. Concurrent

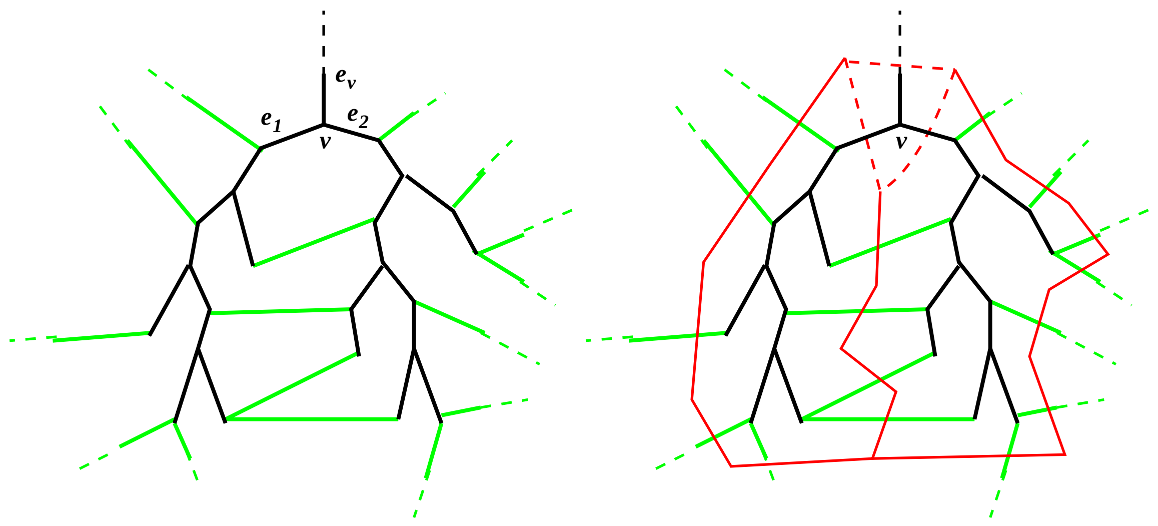


Figure 3. A subgraph arising in the dynamic program. The edge e, the child node v, and the child edges e_1 and e_2 are labeled. The dark edges are tree edges. On the right is the same subgraph with some edges of the dual graph also shown. Note that the edges of $\Gamma(\{$descendents of $e_v\})$ form an elementary cycle in the dual, as do the edges of $\Gamma(\{$descendents of $e_1\})$ and the edges of $\Gamma(\{$desecendents of $e_2\})$.

For the root edge $\hat{e}$ of T (the edge of T incident to r), each edge of $\Gamma_{\hat{H}}(D_{\hat{e}})$ is incident to the root r. It follows that every traversal of $D_{\hat{e}}$ defines a tour of $\hat{H}$ using each dart at most once, and vice versa. By Proposition 9, there is an optimal tour that is noncrossing. Hence

$$\begin{aligned} &\text{OPT}(\hat{H}) \\ &= \min\{M_{\hat{e}}(K) + \frac{1}{2}\text{weight}(K) \ : \ K \text{ a configuration of } C_{\hat{e}}\} \end{aligned}$$

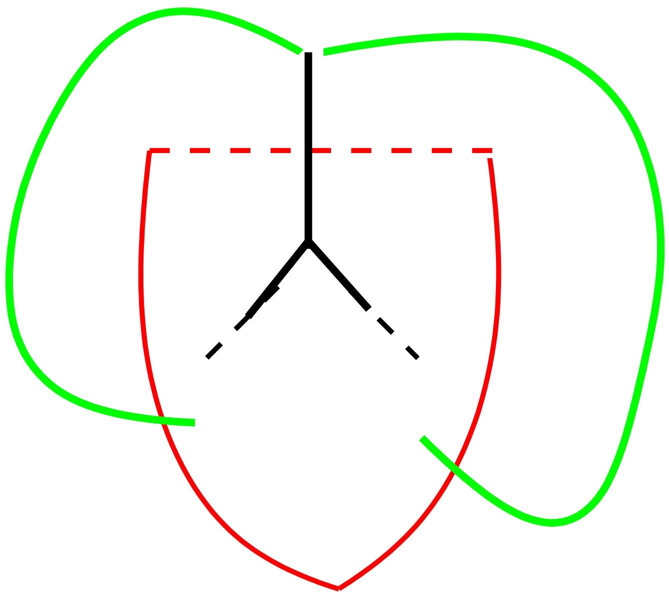
because only half the weight of each edge of K is counted in $M_{\hat{e}}(K)$. Since r has degree at most three, $C_{\hat{e}}$ has $O(1)$ configurations. Thus OPT$(\hat{H})$ can be computed in $O(1)$ time from the table $\text{TAB}_{\hat{e}}$.

6.5 The recurrence relation

Let e be an edge of the tree $\hat{T}$, and let $e_1, \ldots, e_s$ be its child edges ($s \leq 2$). Let $D_0 = \{v_e\}$. For $i = 1, \ldots, s$, let $D_i = D_{e_i}$. Note that D_e is the disjoint union $\bigcup_{i=0}^{s} D_i$.

A traversal $\mathcal{P}$ of D_e *induces* a traversal $\mathcal{P}_i$ of D_i for $i = 0, 1 \ldots, s$ as follows: for each path $P \in \mathcal{P}$, break P into subpaths at the nodes of P that are not in D_i, and retain only those darts d such that at least one endpoint of d is in D_i. The remaining subpaths form a traversal of D_i.

with the appearance of a preliminary version of this paper, Dorn et al. [14] published an extended abstract discussing planar-graph algorithms that also exploited Catalan-type bounds and noncrossing matchings.



Lemma 24 *Let $\mathcal{P}$ be a traversal of D_e, and let $\mathcal{P}_0, \ldots, \mathcal{P}_s$ be the traversals that $\mathcal{P}$ induces for $D_0, \ldots, D_s$. Then*

$$\textit{weight}(\mathcal{P}) = \sum_{i=0}^{s} \textit{weight}(\mathcal{P}_i)$$

Let $K, K_0, K_1, \ldots, K_s$ be configurations of $C_e, C_0, C_1, \ldots, C_s$ respectively. We say that these configurations are *consistent* if every edge occurs zero or two times.

Lemma 25 *For traversals $\mathcal{P}_0, \ldots, \mathcal{P}_s$ of $D_0, \ldots, D_s$, if K is a configuration of C_e such that $K, \kappa(\mathcal{P}_0), \ldots, \kappa(\mathcal{P}_s)$ are consistent then there is a traversal $\mathcal{P}$ of C_e that induces $\mathcal{P}_0, \ldots, \mathcal{P}_s$.*

Proof: By gluing together paths from different $\mathcal{P}_i$'s that have a common edge, one constructs paths whose start and edge edges are in $\Gamma(D_e)$. $\square$

Corollary 26 *For any configuration K of C_e,*

$$M_e(K) = \min\{\sum_{i=0}^{k} M_{D_i}(K_i) \ : \ K, K_0, \ldots, K_s \textit{ are consistent}\}$$

6.6 The dynamic program

We now give a recursive algorithm TSP-DP(e) that for each edge e of T populates the table TAB$_e$.

```
define TSP-DP$(e)$:
1 let $e_1, \ldots, e_s$ be the child edges of $e$ ($s \leq 2$)
2 for $i = 1, \ldots, s$,
3     recursively call TSP-DP$(e_i)$.
4 let $C_0$ :=the cycle in $\hat{H}^*$ formed by $\Gamma_{\hat{H}}(\{v_e\})$
5 initialize each entry of TAB$_e$ to $\infty$.
6 for each tuple $(K_0, K_1, \ldots, K_s)$
              of configurations of $\Gamma(\{v_e\}), \Gamma(D_{e_0}), \ldots, \Gamma(D_{e_s})$
7     let $K$ be the configuration of $\Gamma(D_e)$ induced by $K_0, K_1, \ldots, K_s$
8     TAB$_e[K]$ := $\min\{$TAB$_e[K]$ ,
                         $\sum_{i=0}^{k}$ TAB$_{e_i}[K_i]\}$
```

Corollary 27 (Correctness of TSP-DP) *For each edge e of T, for each noncrossing configuration K of C_e, TAB$_e[K] = M_{D_e}(K)$.*

6.7 Analysis of the dynamic program

In Step 6, each of the cuts $\Gamma(D_{e_1}), \ldots, \Gamma(D_{e_s})$ has size at most $\ell+1$, so has $O(c^\ell)$ configurations for a constant c. The cut $\Gamma(\{v_e\})$ has size at most three, and $s \leq 2$, so the number of tuples in Step 6 is $O(d^\ell)$ for a constant d. Thus each invocation of TSP-DP requires $O(d^\ell)$ time. The number of invocations is $|V(\hat{H})| - 1$, so the entire dynamic program takes time $O(d^\ell |V(\hat{H})|)$. Combined with the reduction of Subsection 6.1, this completes the proof of Theorem 3.

7 Lifting

7.1 Lifting with respect to one compression

We give a procedure LIFT-ONE(G, e, B) that, given an Eulerian bisubgraph B of $G/\{e\}$, returns an Eulerian bisubgraph B' of G such that $B' - \{e\} = B$. (See Figure 4.)

```
define LIFT-ONE$(G, e, B)$:
  if$e$ is a self-loop of $G$ then return $B$
  if the endpoints of $e$ in $G$ have even degree in $B \cup \{e\}$
    then return $B \cup \{e\}$
    else return $B \cup \{e\} \cup \{e\}$
```

Lemma 28 (Correctness of LIFT-ONE) *If B is an Eulerian bisubgraph of $G/\{e\}$ then* LIFT-ONE(G, e, B) *returns an Eulerian bisubgraph of G.*

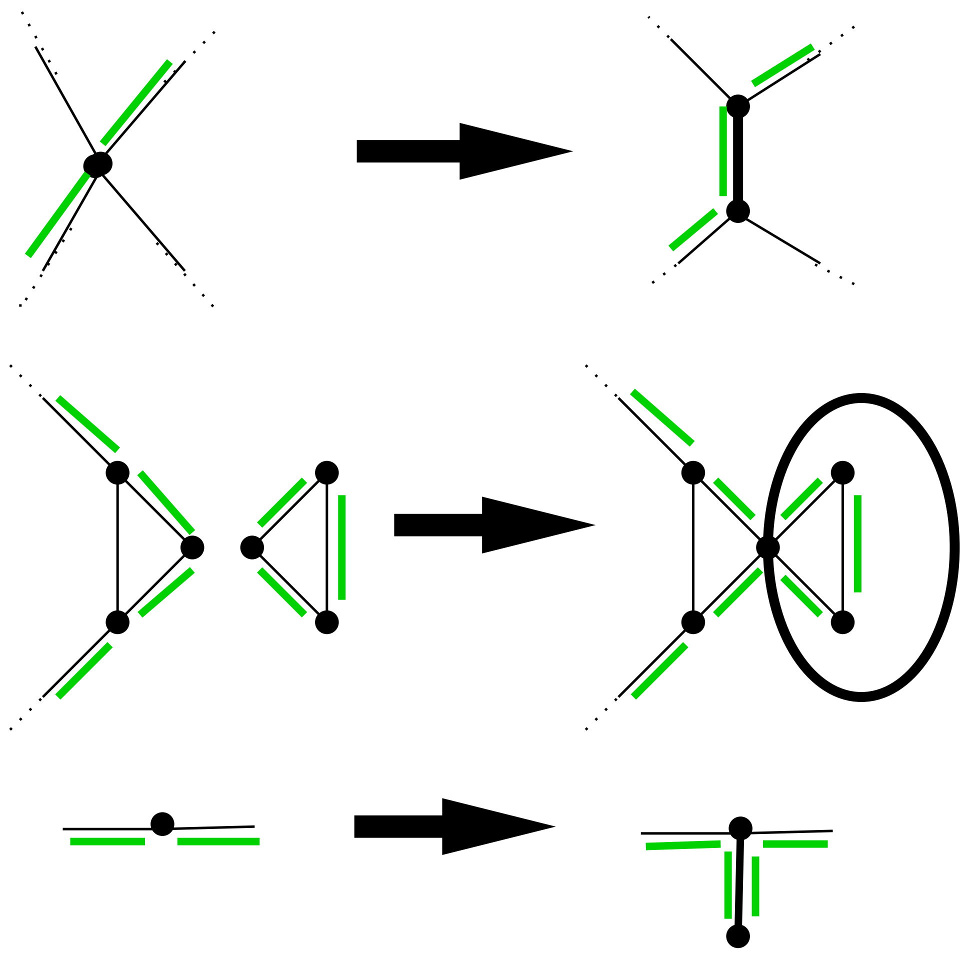


Figure 4. Examples of lifting. The Eulerian bisubgraph is indicated with light lines. In the top diagram, only one copy of the formerly compressed edge (the bold line) must be incorporated. In the second diagram, no copies are needed. In the third diagram, two copies are needed.

Proof: If e is a self-loop of G then $G - \{e\}$ is obtained from $G/\{e\}$ by merging two nodes u and v belonging to different components. Since B consists of a connected Eulerian bisubgraph for each connected component of $G/\{e\}$, u and v have even degree in B (so the merged node has even degree) and B is a connected Eulerian bisubgraph for the combined component.

Suppose e is not a self-loop of G. Then $G/\{e\}$ is obtained from G by contracting e. Since B is an Eulerian bisubgraph of $G/\{e\}$, it must contain at least one edge incident to at least one endpoint of e in G. Therefore $B \cup \{e\}$ connects each connected component of G. Since in $G/\{e\}$ the degree in B of each node is even, in G every node has even degree in B except possibly the endpoints of e. If even the endpoints have even degree then $B \cup \{e\}$ is an Eulerian bisubgraph of G. If not, then both endpoints must have odd degree in B since the sum of their degrees is the degree of the node resulting from contracting e. Hence $B \cup \{e\} \cup \{e\}$ is an Eulerian bisubgraph. $\square$

7.2 Lifting with respect to many compressions

Theorem 4 *There is a linear-time algorithm that, given an Eulerian bisubgraph B of G/S, returns an Eulerian bisubgraph B' of G such that $B' - S = B$.*

We give the procedure below. Its correctness follows by induction from the correctness of LIFT-ONE.

```
define LIFT-MANY$(G, S, B)$:
  if $S = \emptyset$ then return $B$
  else
    let $e$ be an edge of $S$
    return LIFT-MANY$(G, S - \{e\},$ LIFT-ONE$(G/\{e\}, e, B))$
```

8 Low-diameter planar graphs have small branch-width

We restate and prove Lemma 20. The proof uses the ideas used in formulating the dynamic program.
Lemma 20 *Let G be a planar graph and let T be a spanning tree of G such that every simple path in T has length at most ℓ. Then the branch-width of the dual G^* is at most $\ell + 2$.*
Proof: Let $\hat{G}$ be the planar embedded graph obtained from G by adding artificial edges until each face has at most three edges. Note that T is a spanning tree of $\hat{G}$. Let T^* be the spanning tree of $\hat{G}^*$ consisting of the edges of $\hat{G}^*$ not in T. Since every face of $\hat{G}$ has at most three edges, $\hat{G}^*$ has degree at most three, so T^* has degree at most three. Root T^* at a leaf r. For each node v of $\hat{G}^*$ other than r, let X_v denote the set of descendents of v in T^*, and let Y_v denote the set of proper descendents of v (i.e. excluding v itself). Let $\mathcal{C} = \{X_v : v \neq r\} \cup \{Y_v : v \neq r\} \cup \{\{v\} : v \neq r\}$. Then $\mathcal{C}$ is a carving of $\hat{G}^*$'s *node-set* $V(\hat{G}^*)$.

For any $v \neq r$, let e_v denote the edge of T^* between v and its parent. Then e is not an edge of T.

By Corollary 4,

$$\Gamma_{G^*}(X_v) = E(\text{elementary cycle of } e \text{ with respect to } T)$$

The elementary cycle of e with respect to T consists of e together with the simple path in T between e's endpoints. Since every simple path in T has length at most ℓ, it follows that $|\Gamma_{\hat{G}^*}(X_v)| \leq \ell + 1$. (See Figure 3.)

Let D be the set of edges between v and Y_v. Because the other endpoint of e_v is the parent of v, $e_v \notin D$. Because v has degree at most three, therefore, $|D| \leq 2$. Note that $\Gamma_{\hat{G}^*}(Y_v) = \Gamma_{G^*}(X_v) - \{e_v\} \cup \cup D$, so $|\Gamma_{\hat{G}^*}(Y_v)| \leq \ell + 2$.

Next, we define a carving of $\hat{G}^*$'s edge-set $E(G)$

$$\mathcal{C}' = \{E(G[X]) \ : X \in \mathcal{C}\}$$

where $G[X]$ denotes the subgraph of G induced by the nodes of X.

For each $X \in \mathcal{C}$, a node v of G is in $\partial(E(G[X]))$ only if $v \in X$ and there is an edge $e \in \Gamma(X)$ one of whose endpoints is v. Thus $|\partial(E(G[X]))| \leq |\Gamma(X)|$. This proves that $\mathcal{C}'$ has width at most $\ell + 2$.

We have shown that $\hat{G}^*$ has branch-width at most $\ell + 2$. It is easy to see (and shown in [28]) that edge contraction does not increase branch-width. Since G^* can be obtained from $\hat{G}^*$ by contracting the artificial edges, it follows that the branchwidth of G^* is at most $\ell + 2$. □

The proof of the lemma makes clear that a tree representation of the carving of $E(G^*)$ can be obtained in linear time from G and T.

9 Final remarks

The framework we have presented can be used to obtain approximation schemes for other problems. For example, a similar approach yields a linear-time approximation scheme for the following problem.

- input: a connected planar embedded graph G, and an edge-weight assignment
- output: a minimum-weight multisubgraph of G in which there are two edge-disjoint paths between every pair of nodes.

In particular, the same spanner result can be used.

Another related problem is TSP in a *sub-metric space* of that defined by a planar graph with edge-weights, i.e. the following problem.

- input: a connected planar embedded graph G, an edge-weight assignment, and a subset R of nodes,
- output: a minimum-weight closed walk visiting all nodes in R.

Applying the framework to this problem requires a new spanner result: that there exists a subgraph of G that approximately preserves all distances between nodes of R and that has weight $O(1)$ times that of a minimum-weight Steiner tree for R. We have recently obtained such a result.

Acknowledgements

Claire Kenyon and Mikkel Thorup contributed greatly to the quality of the presentation through their criticisms and suggestions. Thanks also to Glencora Borradaile for helpful discussions and suggestions.

References

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, L. Soares, “On sparse spanners of weighted graphs,” *Discrete and Computational Geometry*, **9**:1, 1993.

[2] S. Arora, “Polynomial-time approximation schemes for Euclidean TSP and other geometric problems,” *Journal of the ACM* **45**:753-782, 1998.

[3] S. Arora, “Polynomial-time approximation schemes for Euclidean TSP and other geometric problems,” *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pp. 2-12, 1996.

[4] S. Arora, “Nearly linear time approximation schemes for Euclidean TSP and other geometric problems,” *Proceedings of the 38th Annual IEEE Foundations of Computer Science*, pp. 554-563, 1997.

[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] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM* **41**:153–180, 1994.

[7] Andr Berger, Artur Czumaj, Michelangelo Grigni, and Hairong Zhao. Approximate minimum 2-connected subgraphs in weighted planar graphs. *Proceedings of the 13th Annual European Symposium on Algorithms*, pp. 472-483, 2005.

[8] H. L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, *SIAM Journal on Computing* **25**:1305–1317, 1996.

[9] D. R. Cheriton, R. E. Tarjan, “Finding minimum spanning trees,” *SIAM Journal on Computing* **5**: 724-742, 1976.

[10] N. Christofides, “Worst-case analysis of a new heuristic for the traveling salesman problem,” In J.F. Traub, editor, *Symposium on new directions and recent results in algorithms and complexity*, p. 441. Academic Press, NY, 1976.

[11] , W. J. Cook and P. D. Seymour, “Tour Merging via Branch-decomposition,” *INFORMS Journal on Computing*, **15**:233–248, 2003.

[12] A. Czumaj, M. Grigni, P. Sissokho, and H. Zhao, “Approximation schems for minimum 2-edge-connected and biconnected subgraphs in planar graphs,” *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms,* pp. 489–498, 2004.

[13] E. D. Demain, 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.

[14] F. Dorn, E. Penninkx, H. L. Bodlaender, F. V. Fomin,”Efficient exact algorithms on planar graphs: exploiting sphere cut branch decompositions,” *Proceedings of the 13th Annual European Symposium on Algorithms*, pp. 95–106, 2005

[15] D Eppstein, “ Subgraph isomorphism in planar graphs and related problems,” *Journal of Graph Algorithms and Applications* **3**:1–27, 1999.

[16] 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.

[17] M. Grigni and P. Sissokho, “Light spanners and approximate TSP,” *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms* , pp. 852–857, 2002.

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

[19] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, “Faster shortest-path algorithms for planar graphs,” *Journal of Computer and System Sciences* **55**:3-23, 1997.

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

[21] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, *The traveling salesman problem*, John Wiley, 1985.

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

[23] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences* **43**:425–440, 1991.

[24] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research* **18**:1–11, 1993.

[25] 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.

[26] N. Robertson and P. D. Seymour, “Graph Minor III. Planar Tree-Width,” *Journal of Combinatorial Theory, Series B*, **36**:49–63, 1984.

[27] N. Robertson and P. D. Seymour, “Graph minors IV: Tree-width and well-quasi-ordering,” *Journal of Combinatorial Theory, Series B* **48**::227-254, 1990.

[28] N. Robertson and P. D. Seymour, “Graph minors X. Obstructions to tree-decomposition,” *Journal of Combinatorial Theory, Series B,* **52**:153–190, 1991.

[29] P. D. Seymour and R. Thomas, “Call routing and the ratcatcher,” *Combinatorica* **14**(2):217–241, 1994.