A Polynomial-time Approximation Scheme for Planar Multiway Cut

MohammadHossein Bateni* MohammadTaghi Hajiaghayi† Philip N. Klein‡
Claire Mathieu§

Abstract

Given an undirected graph with edge lengths and a subset of nodes (called the *terminals*), the *multiway cut* (also called the *multi-terminal cut*) problem asks for a subset of edges, with minimum total length, whose removal disconnects each terminal from all others. The problem generalizes *minimum s-t cut*, but is NP-hard for planar graphs and APX-hard for general graphs [11]. In this paper, we present a PTAS for multiway cut on planar graphs.

1 Introduction

In the *multiway cut problem* (a.k.a. *multi-terminal cut problem*), given an undirected graph with edge lengths and a subset of nodes called the *terminals*, the goal is to disconnect the terminals from one another using a subset of edges of minimum total length. With k denoting the cardinality of the set of terminals, the problem is sometimes also called the *k-terminal cut problem* or the *k-way cut problem*.

It is a natural problem: it generalizes the problem of finding a minimum-length st-cut. A variant was first proposed in T. C. Hu's 1969 book [15]. The study of its computational complexity was inaugurated in 1983 by Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis [11][1]. Their results, already highlighting the case of planar graphs, have guided the agenda for subsequent research:

1. For general graphs, there is a simple 2-approximation algorithm disconnecting each terminal from the others by a minimum cut, but for any fixed $k \geq 3$, the problem is APX-hard, and so no polynomial-time approximation scheme (PTAS) exists if P $\neq$ NP.

2. For planar graphs, the problem can be solved in polynomial time for fixed k but is NP-hard when k is unbounded.

Result 1 led to a sequence of constant-factor approximation algorithms with improved approximation factors; see, e.g., [8, 10, 17]. Result 2 spawned papers giving improved running times for the case of planar graphs and fixed k; see, e.g., [3, 9, 14, 16]. In this paper, we provide a result that complements Result 2; we show there is a polynomial-time approximation scheme for multiway cut on planar graphs.

THEOREM 1.1. *There is a polynomial-time approximation scheme (PTAS) for the multiway cut problem on planar graphs. Its running time is $O(f(\epsilon)n^c)$, where $f(\epsilon)$ is a function of ϵ independent of n and c is an absolute constant independent of ϵ.*

Theorem 1.1 is in the continuation of a sequence of results designing PTASes for planar graph instances of progressively harder optimization problems with connectivity constraints: TSP [18, 19, 20], Steiner tree [4, 5, 7], and Steiner forest [2]. Our new algorithm builds on the brick decomposition technique from [5] (which in turn builds on [19]) and the prize-collecting-clustering technique from [2], as well as a technique for finding short cycles enclosing prescribed amounts of weight from a paper by Park and Phillips [21].

In [18, 20], Klein stated a strategy in two forms, a primal form and a dual form. In the dual form, which applies here, the strategy is as follows: Step 1: contract edges from the input graph G_{in} to get a graph

*Google Research, New York, NY 10011; The work was done while the author was with Department of Computer Science, Princeton University and with Center for Computational Intractability, Princeton, NJ 08540; Email: mbateni@cs.princeton.edu.

†Supported in part by a Google Faculty Research Award, an ONR Young Investigator Award, an NSF CAREER Award and a DARPA BAA grant. Department of Computer Science, University of Maryland at College Park, USA; Email: hajiagha@cs.umd.edu. The author is also with ATT Labs–Research, USA.

‡Computer Science Department, Brown University, Providence, RI 02912; Email: klein@brown.edu. Research supported in part by NSF Grant CCF-0964037

§Computer Science Department, Brown University, Providence, RI 02912; Email: claire@cs.brown.edu. Research supported in part by NSF Grant CCF-0964037

[1]The work was first known in an unpublished but widely circulated extended abstract. Their complete paper was published in 1994.

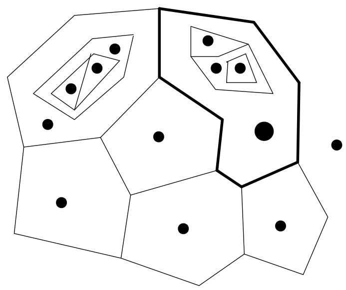

Figure 1: This figure shows a multiway-cut solution in the planar dual. The line segments are the edges of the planar dual that belong to the solution, and the black dots denote the terminals (which are faces in the dual). A feasible solution has the property that each region contains at most one terminal. Note that the solution need not be connected, that there is arbitrary nesting of connected components, and that regions need not be simply connected (i.e. a region's boundary need not be connected). However, with respect to a designated infinite face, each region except the outermost does have a simple enclosing cycle; for the terminal t inhabiting the region, this cycle is the minimally enclosing cycle in the solution that encloses t. For example, for the terminal represented by the oversize dot, the minimally enclosing cycle is indicated with thicker line segments.

$\widehat{G_{in}}$ whose length is $O(OPT)$ and that approximately preserves OPT. Step 2: for a given constant ζ, find a set of edges of weight at most $\frac{1}{\zeta}\text{length}(H)$ whose deletion from $\widehat{G_{in}}$ yields a graph of branchwidth $O(\zeta)$. Step 3: Solve the problem in the bounded-branchwidth graph. Step 4: Lift the solution to the original graph, incorporating some of the edges deleted. The value of ζ is chosen so that the length of the deleted edges is a small fraction of the length of $\widehat{G_{in}}$ and therefore an ϵ fraction of OPT.

This strategy has been used for the aforementioned optimization problems. The hardest part for multiterminal cut (as for most problems addressed with the framework) is Step 1. The graph $\widehat{G_{in}}$ obtained in Step 1 is called a *spanner* by analogy to distance spanners. Our main contribution in this paper is a spanner construction for multiterminal cut.

The starting point for our spanner construction is that, in the planar dual, a multiterminal cut resembles a Steiner tree in structure. This suggests that the techniques used for Steiner tree (*brick decompositions* and *portal-respecting solutions*) could be used here. (These techniques are summarized in Section 3.4.) However, as shown in Figure 1, the solution in the dual need not be connected. In this way, the problem is similar to Steiner forest, in which the solution need not be connected. This suggests we employ a method used for Steiner forest, *prize-collecting clustering.* This method, given a graph with edge-lengths and vertex-potentials, finds a forest F whose length is at most twice the sum of potentials and such that any low-cost forest L does not connect distinct trees in F, if we disregard a set of vertices whose total potential is at most the length of L. (This method is summarized in Section 3.3.)

There are two additional difficulties, however. In Steiner tree and Steiner forest, an instance specifies a set of vertices, the terminals, that must be connected up by the edges forming the solution network. In multiterminal cut, an instance specifies terminals, but these must be *separated* by the network. In particular, in the planar dual, where terminals become faces, the terminals can be quite far from the edges forming the solution. To address this problem, we introduce a technique of choosing, for each terminal t, a set of simple cycles in the dual that enclose t, such that at least one of these cycles intersects the part of a near-optimal solution that separates t from the infinite face.

Another difficulty is as follows. (Again, refer to Figure 1.) Consider a multiterminal cut solution in the dual, and consider a connected component K. This component serves to separate some terminals $t_1, \ldots, t_p$ from each other. However, one of these terminals is not enclosed in K. It lies outside K, and so must in turn be separated from other terminals by another connected component. Thus the connected components cannot be handled independently from each other. To address this, we give an algorithm to construct a subgraph, called the *skeleton.* The nesting structure of connected components of the skeleton approximates the nesting structure of connected components of a near-optimal solution. This enables us to construct the spanner.

2 Overview

We denote the input graph by G_{in}. The set of terminals is denoted T, and the assignment of lengths to edges is $\text{length}(\cdot)$. We give an algorithm that, for a given error tolerance ϵ, finds a multiterminal cut of length at most $1 + c\epsilon$ times optimal, where c is a constant.

For notational simplicity, we consider T, $\text{length}(\cdot)$, and ϵ as global variables so we don't have to pass them as arguments to the procedures we define. For a graph G derived from G_{in} by deletions and contractions, we use the notation $\mathsf{OPT}(G)$ to refer to the minimum length of a multiterminal cut in G separating the terminals $T \cap V(G)$.

The algorithm for finding an approximate multiway cut follows the strategy discussed in the introduction. In Line 1, the graph $\widehat{G_{in}}$ is obtained from G_{in} by edge contractions but still contains all of the terminals. Therefore the edges forming a multiterminal cut in $\widehat{G_{in}}$ also form a multiterminal cut in G_{in}. By averaging, in

Algorithm 1 MultiwayCut $(G_{in}, T_{in}, \text{length}(\cdot))$

Input: planar graph G_{in}, terminals T, length assignment length$(\cdot)$

Output: a $(1 + (c + 1)\,\epsilon)$-approximate multiterminal cut where c is a constant

1: $\widehat{G_{in}} \leftarrow$ MainSpanner(G_{in})
2: **comment:** $\text{length}(\widehat{G_{in}}) \leq f(\epsilon)\mathsf{OPT}(G_{in})$ and $\mathsf{OPT}(\widehat{G_{in}}) \leq (1 + c\epsilon)\mathsf{OPT}(G_{in})$
3: $r \leftarrow$ some vertex of $\widehat{G_{in}}$
4: $\zeta = \epsilon^{-1} f(\epsilon)$
5: for $i = 0, \ldots, r - 1$, let E_i be the set of edges e of $\widehat{G_{in}}$ such that breadth-first search distance from r to e is congruent mod ζ to i.
6: let E_{i^*} be the set of minimum length.
7: construct a branch decomposition of $\widehat{G_{in}} - E_{i^*}$ of width 2ζ
8: $M \leftarrow$ optimal multiterminal cut for $\widehat{G_{in}} - E_{i^*}$
9: **return** $E_{i^*} \cup M$

Line 6,

$$\begin{aligned} \text{length}(E_{i^*}) &\leq \frac{1}{\zeta}\text{length}(\widehat{G_{in}}) \\ &\leq \epsilon\,\mathsf{OPT}(G_{in}) \end{aligned}$$

Since

$$\begin{aligned} \mathsf{OPT}(\widehat{G_{in}} - E_{i^*}) &\leq \mathsf{OPT}(\widehat{G_{in}}) \\ &\leq (1 + c'\epsilon)\mathsf{OPT}(G_{in}) \end{aligned}$$

it follows that the solution $E_{i^*} \cup M$ returned in Line 9 has length at most $(1 + c'\epsilon)\mathsf{OPT}(G_{in}) + \epsilon\,\mathsf{OPT}(G_{in})$.

We briefly review the notion of *branchwidth*. Two sets A and B *cross* if $A - B, B - A, A \cap B$ are nonempty. A carving $\mathcal{C}$ of a ground set S is a maximal collection of noncrossing subsets of S. A *branch decomposition* of a graph is a carving $\mathcal{C}$ of the edges of the graph. The *boundary* of a set A of edges is the set of A vertex v is on the *boundary* of a set A if the edges in A that are incident to v form a nonempty proper subset of the edges incident to v. The *width* of a branch-decomposition $\mathcal{C}$ is the maximum size of the boundary of a set $A \in \mathcal{C}$.

The argument that, in Line 7, $\widehat{G_{in}} - E_{i^*}$ has bounded branchwidth (or treewidth) dates back to the work of Baker [1].[2] It is a simple exercise to show that, given an m-edge graph G and a branch-decomposition of width w, an optimal multiterminal cut can be found in $2^{O(w)}m$ time.

In order to prove Theorem 1.1, it remains to give the algorithm used in Line 1.

Theorem 2.1. *There is a constant d such that, for any constant $\epsilon > 0$, there is an $O(n^d)$-time algorithm that, given an instance $(G_{in}, T, \mathit{length}(\cdot))$ of multiterminal cut, constructs a subgraph $\widehat{G_{in}}$ of G_{in} such that $\mathit{length}(\widehat{G_{in}}) \leq 2^{poly(1/\epsilon)}\mathsf{OPT}(G_{in})$ and $\mathsf{OPT}(\widehat{G_{in}}) \leq (1 + c\epsilon)\mathsf{OPT}(G_{in})$*

Outline of the proof of Theorem 2.1 The rest of the paper is devoted to proving Theorem 2.1. Here we provide an overview of the algorithm.

Each terminal t is assigned a weight equal to the minimum length of a cut separating t from all other terminals. The procedure MainSpanner(G_{in}) uses short simple cuts to decompose G_{in} into graphs G in which (almost) any cut $\delta(S)$ has length at least ϵ times the total weight of S. For each such graph G, a subgraph H (the *skeleton*) is computed, and a spanner for G is constructed from G and its skeleton H. Finally, MainSpanner returns the union of these spanners with the short simple cuts used to decompose G_{in}.

The procedure Skeleton for finding the skeleton operates on the planar dual G^* of G, as follows.

- For each terminal t, the algorithm selects several cycles in the planar dual (cuts in the primal) that enclose t, such that at least one of those cycles intersects the component that immediately encloses t in a near-optimal solution.
- The algorithm iteratively adds paths we call *ears* to each face of each connected component of the skeleton so far. An ear must separate two terminals each of weight at least ϵ^3 times the length of the ear.
- The algorithm runs *prize-collecting clustering* (described in Section 3.3) to augment the skeleton so far with some additional edges.

Now we describe the construction of the spanner for G. It too operates on the planar dual G^*.

- For each connected component K of the skeleton, the algorithm constructs a *brick decomposition* (described in Section 3.4) starting from K. This defines a subgraph M of G that includes K. For each face of M, the part of G embedded within the face is called a *brick*.
- For each brick, the algorithm designates as *portals* a constant number of evenly spaced vertices on the boundary of the brick (the face of M). The algorithm identifies a constant number of important

[2] Baker's work predated the notions of branchwidth or treewidth were formulated; see [12, 20] for arguments using these notions.

terminals in the brick. computes minimum-length *portal-respecting partial solutions*, subgraphs of the brick that separate parts of the brick from each other and from a single important terminal. Because the number of portals is constant and the number of important terminals is constant, only a (large) constant number of partial solutions need to be computed.

- The spanner for G is defined to be the the union of all the min-cuts of ther terminals, all the brick decompositions (which include all the skeletons), and all the portal-respecting partial solutions.

3 Background

3.1 Terminology We use the notation $G[V']$ for the subgraph of G induced by a subset V' of the vertex set of G.

In a graph G, the *cut* defined by a set S of vertices, denoted by $\delta_G(S)$, is the set of edges having one endpoint in S and one endpoint not in S. It is *simple* if $G[S]$ and $G[V-S]$ are both connected graphs.

For every connected planar embedded graph G, there is another planar embedded graph G^*, called the *planar dual* of G. The vertices of G^* are the faces of G, and vice versa. For each edge e of G, there is an edge in G^* (which we also call e) between the two faces bordering e in G. A classical result in graph theory states that, for a planar graph G, a subset of edges form a simple cut in G if the same edges form a simple cycle in G^*. Contracting a (non-self-loop) edge in G corresponds to deleting the edge in G^* and vice versa.

We can think of a planar graph being embedded either on a plane or on a sphere. On the plane, there is one face that is infinite. We prefer to think of the graph being embedded on the sphere, so the choice of the "infinite" face is arbitrary. With respect to a designated infinite face, we say a simple cycle C in G^* *encloses* a face of G^* if in G the edges of C separate f from the infinite face. We say C encloses a vertex or edge if C encloses some face incident to the vertex or edge. We say C encloses a subgraph if C encloses every edge of the subgraph. We say C *strictly* encloses the subgraph if in addition no vertex of the subgraph lies on C. We say a subgraph H encloses a subgraph H' if some cycle of H encloses H'. The *outer boundary* of H is the set of edges that are part of H and not strictly enclosed by H.

3.2 Finding short cycles and paths Let G be an undirected planar embedded graph with edge-lengths and face-weights. Let T be a spanning tree of G. Park and Phillips [21] give a technique for turning weight enclosed by a cycle into total weight assigned to the cycle. Root T at a node r. Each edge of G corresponds to two oppositely directed *darts*. Assign weights to the darts as follows. The weight of a dart belonging to an edge of T is zero. For a dart uv not in T, there is a unique simple cycle consisting of uv and the u-to-v path in T, called the *fundamental cycle* of uv with respect to T. If the fundamental cycle of uv is clockwise, the weight of uv is defined to be the sum of the weights of the faces enclosed by the elementary cycle, and the weight of vu, the oppositely directed dart, is the negative of this sum.

It is easy to verify that, for any simple clockwise cycle C, the weight of the darts forming C equals the weight enclosed by C (and the weight of the darts forming the reverse cycle is the negative of the weight enclosed).

3.3 Prize-collecting clustering In our algorithm, we use the PC-CLUSTERING algorithm of Bateni et al. [2] as a subroutine. Theorem 3.1 summarizes its guarantees, which we will use in the following way. There is a cost weight(v) associated with *ignoring* each vertex v (vertex v might be itself a supervertex obtained by contracting a connected subgraph). We let $\phi(v) = \epsilon^{-2}\text{weight}(v)$ for all vertices v, invoke the procedure, and apply the theorem on a near-optimal solution L to obtain Q. Then, the total cost of Q is $\sum_{v\in Q}\text{weight}(v) \leq \epsilon^2\text{length}(L) \approx \epsilon^2\mathsf{OPT}$. Thus, all these vertices are going to be ignored from consideration, paying only a negligible cost.

THEOREM 3.1. (PRIZE-COLLECTING) *Let G be a graph with edge lengths such that each vertex v has a potential $\phi(v)$, and let H be the subgraph of G output by the* PC-CLUSTERING *algorithm executed on (G,ϕ). Then*

1. *length$(H) \leq 2\sum_v \phi(v)$.*
2. *For any subgraph L of G, there is a set Q of vertices such that*
   (a) $\sum_{v\in Q}\phi(v) \leq$ length(L); and
   (b) If two vertices $v_1, v_2 \notin Q$ are connected by L, they are in the same connected component of H.

Proof. (See Figure 4 for an example application). The reader is referred to [2] for details of the algorithm itself.

The PC-CLUSTERING algorithm builds a forest F, and produces a vector y satisfying

$$\text{(3.1)} \quad \sum_{S:e\in\delta(S)}\sum_{v\in S} y_{S,v} \leq c_e \qquad \forall e\in E$$

$$\text{(3.2)} \quad \sum_{S\ni v} y_{S,v} = \phi(v) \qquad \forall v\in V$$

$$\text{(3.3)} \quad y_{S,v} \geq 0 \qquad \forall v\in S\subseteq V.$$

The analysis takes advantage of the connection between F and y. Consider a topological structure in which vertices of the graph are represented by points, and each edge is a curve connecting its endpoints whose length is equal to the weight of the edge. We assume that each vertex v has a unique color. The algorithm paints by color v a connected portion with length $y_{S,v}$ of all the edges in $\delta(S)$. In particular, each edge e gets exactly $\sum_{C:e\in\delta(S)} y_{S,v}$ units of color v. Property 1 of the statement of the theorem follows directly from the following lemma.

LEMMA 3.1. ([2]) *The length of F is at most $2\sum_v \phi(v)$.*

In the rest of the proof, we establish the second property of the statement. We say a graph $G'(V, E')$ *exhausts* a color u if and only if $E' \cap \delta(S) \neq \emptyset$ for any $S : y_{S,u} > 0$. Note that this does not imply that all edges with color u are part of E'.

LEMMA 3.2. ([2]) *If a subgraph L of G connects two vertices u_1, u_2 from different components of F, then L exhausts the color corresponding to at least one of u_1 and u_2.*

We can also relate the length of a subgraph to the potential value of the colors it exhausts.

LEMMA 3.3. ([2]) *Let X be the set of colors exhausted by a subgraph L of G. Then length(L) is at least $\sum_{v\in X} \phi(v)$.*

We add to Q any vertex whose color is exhausted by L. Lemma 3.3 gives Property 2a. For Property 2b, suppose L connects two vertices u_1, u_2 that are in different connected components of H. By Lemma 3.2, L exhausts the color of at least one of u_1, u_2, so it is placed in Q. □

3.4 Brick decomposition In our algorithm, we use, as a subroutine, the brick decomposition algorithm of [6] based on the spanner construction of [19]. Given a connected subgraph K of a planar embedded graph G and given parameters $\epsilon > 0, \kappa > 1$, there is an $O(n \log n)$ algorithm to compute a connected subgraph M of G, called the *mortar graph*. For each face f of M, the subgraph of G enclosed by ∂f (including the boundary) is called a *brick*.

For any $\epsilon > 0$ and $\kappa > 1$, there is an $O(n \log n)$ algorithm that, given a planar embedded graph G and a connected subgraph K of G, finds a connected subgraph M of G that contains K. For each face of M, the subgraph of G enclosed by the boundary of the face is called a *brick*. The brick includes the boundary of the face, which is called the boundary of the brick. The boundary of a brick B consists of four paths, $W_B \cup S_B \cup E_B \cup N_B$ (west, south, east, north).

The brick decomposition satisfies two length properties: the length of M is a constant times the length of K:

$$\text{length}(M) \leq (1 + 1/\epsilon + 1/(\kappa\epsilon^2))\text{length}(K) \tag{3.4}$$

and the east and west boundaries represent a small fraction of this:

$$\sum_B (\text{length}(W_B) + \text{length}(E_B)) \leq \frac{1}{\epsilon\kappa}\text{length}(M) \tag{3.5}$$

Now we come to the most significant property of the brick decomposition. For a brick B and a subgraph F of B, a *joining vertex* of F with the boundary of B is a vertex of the boundary that is the endpoint of an edge of F not in the boundary. The theorem stated below is a slight refinement of Theorem 10.7 of [7]. The main point is that, given a subgraph F of a brick, a replacement subgraph (not too much longer than F) spans the same boundary vertices but has few joining vertices.

THEOREM 3.2. *Let B be a brick with boundary $W \cup S \cup E \cup N$, let F be a set of edges in B, and let $\mathcal{U} = \{u_0, u_1\}$ be a set of at most two nodes of F. Then there exists a forest F' of B with the following properties:*

- *F' has $O(\epsilon^{-2.5}\kappa)$ joining vertices.*
- *If two vertices of $\mathcal{U} \cup \{$boundary of $B\}$ are connected in F, then they are also connected in F'.*
- *length$(F') \leq (1 + \epsilon)$length(F) + length$(E \cup W)$*
- *All edges of F' are in the subgraph of B enclosed by $F \cup N_F \cup S_F$, where N_F (resp. S_F) denotes the subpath of N (resp. S) spanned by F.*

4 Simplifying the problem

In this section, we give the procedure MAINSPANNER(G_{in}) for finding a spanner for G_{in}. The procedure computes a *weight* for each vertex, and then uses small simple cuts to decompose G_{in} into smaller graphs in which there is no simple cut whose length is small compared to the weight on one side of the cut. Combining spanners for these smaller graphs with the small simple cuts used in the decomposition yields a spanner for G_{in}.

4.1 Vertex weights For each terminal $t \in T$, the algorithm computes the minimum cut $\mathsf{mincut}(t)$ separating t from $T - \{t\}$ (the rest of the terminals). The algorithm assigns weights to the vertices. For each terminal t, weight$(t) \leftarrow$ length$(\mathsf{mincut}(t))$, and for each nonterminal vertex v, weight$(v) \leftarrow 0$. Let W_{in} denote the sum of weights. For a set S of vertices, weight(S) denotes $\sum_{v\in S}$ weight(v).

LEMMA 4.1. ([11]) $\mathsf{OPT}(G_{in}) \leq W_{in} \leq 2\mathsf{OPT}(G_{in})$

The vertex weights will not change for the duration of the algorithm, and so we consider the weight assignment weight$(\cdot)$ as a global.

4.2 Graphs without short simple cuts Let G be a graph with edge-lengths and vertex-weights. Let v_∞ be a nonterminal vertex. For a vertex subset $S \subsetneq V - \{v_\infty\}$, the *length-weight ratio* is defined to be $\text{length}(\delta_G(S))/\text{weight}(S)$. We say G is *ϵ-short-cut-free with respect to v_∞* if the length-weight ratio of every $S \subsetneq V - \{v_\infty\}$ is at least ϵ.

The algorithm now essentially reduces the problem of finding a spanner to the ϵ-short-cut-free case. To do this, it repeatedly looks for a set S with a small length-weight ratio, finds a spanner for the S part of the graph (loosely speaking), and chops S out of the graph. The overall spanner is the union of these spanners together with the small length-weight-ratio cuts $\delta(S)$. The pseudocode below specifies this more precisely, using subroutines Skeleton and Spanner described later.

Algorithm 2 MainSpanner(G_{in})

Input: planar graph G_{in} with edge-lengths length$(\cdot)$, vertex-weights weight$(\cdot)$, and terminals T

Output: a spanner $\widehat{G_{in}}$ for the Multiway Cut instance

1: Initialize $G_0 \leftarrow G_{in}$, $\widehat{G_{in}} \leftarrow \emptyset$
2: $v_0 \leftarrow$ some nonterminal vertex.
3: **while** there exists $S \subset V(G_0) - \{v_0\}$ whose length-weight ratio is less than ϵ **do**
4: let S be a minimal such set such that $\delta_{G_0}(S)$ is a *simple* cut
5: Let G be the graph obtained from G_0 by merging all vertices not in S to a single vertex v_∞
6: $\widehat{G} \leftarrow$ Spanner(G, v_∞, Skeleton(G, v_∞))
7: $\widehat{G_{in}} \leftarrow \widehat{G_{in}} \cup \widehat{G} \cup \delta(S)$
8: Delete vertices of S from G_0
9: $G \leftarrow G_0$ and $v_\infty \leftarrow v_0$
10: $\widehat{G_{in}} \leftarrow \widehat{G} \cup$ Spanner(G, v_∞, Skeleton(G, v_∞))
11: **return** $\widehat{G_{in}}$

It is easy to see that the existence of a set $S \subset V(G_0) - \{v_\infty\}$ with length-weight ratio less than ϵ implies the existence of such a set with the additional restriction that $\delta_{G_0}(S)$ is a simple cut. This justifies imposing the restriction in Line 4. The algorithm for Line 4 uses planarity; we describe it in Section 4.4.

4.3 Correctness of MainSpanner$(\cdot)$ Suppose MainSpanner runs for k iterations, and, for $i = 1, \ldots, k$, in the i^{th} iteration, S_i is the set chosen in Line 4, $C_i = \delta_G(S_i)$ is the corresponding simple cut, and G_i and $v_{i,\infty}$ are the graph G and the vertex v_∞ obtained in Line 5. Let G_{k+1} and $v_{k+1,\infty}$ be the graph and vertex assigned to G and v_∞ in Line 9, after the k iterations.

Lemma 4.2. *$\sum_{i=1}^{k}$ length$(C_i) \le \epsilon W_{in}$*

Proof. In each iteration, $\text{length}(\delta_{G_0}(S)) < \epsilon\,\text{weight}(S)$. At the end of each iteration, the vertices of S are removed from G_0, so the sum of lengths of the cuts is less than ϵ times W_{in}. □

Lemma 4.3. *For $i = 1, \ldots, k+1$, G_i is ϵ-short-cut-free with respect to $v_{i,\infty}$.*

Proof. There are two cases. Suppose $i \ne k+1$. In this case, $V(G_i) - \{v_{i,\infty}\} = S_i$, so the property holds by the minimality of S_i.

Suppose $i = k+1$. In this case, due to the **while** condition, there is no subset S of $V(G_{k+1}) - \{v_\infty\}$ whose length-weight ratio is less than ϵ □

Having established Lemma 4.3, we avoid undoing it by not recomputing the weights for the terminals in each graph G_i. Let $W_i = \sum_{v \in V(G_i)} \text{weight}(v)$. Since we don't recompute the weights for G_i, W_i is not necessarily a lower bound on $2\mathsf{OPT}(G_i, \text{length}(\cdot))$. However, since each terminal appears in only one graph G_i, $\sum_{i=1}^{k+1} W_i$ is a lower bound on $2\mathsf{OPT}(G_{in}, \text{length}(\cdot))$. In finding the spanner for each subgraph G_i, therefore, we can tolerate an error of $O(\epsilon)W_i$.

Lemma 4.4. *Assume that the result of calling* Spanner *on G_i is a subgraph $\widehat{G_i}$ of G_i such that*

- *length$(\widehat{G_i}) \le 2^{poly(1/\epsilon)} W_i$, and*
- *$\widehat{G_i}$ contains a multiway cut for G_i whose length is at most $\mathsf{OPT}(G_i) + c\epsilon W_i$*

Then MainSpanner *returns a subgraph $\widehat{G_{in}}$ of G_{in} such that*

- *length$(\widehat{G_{in}}) \le 2^{poly(1/\epsilon)} \mathsf{OPT}(G_{in})$, and*
- *$\widehat{G_{in}}$ contains a multiway cut for G_{in} whose length is at most $(1 + c'\epsilon)\mathsf{OPT}(G_{in})$*

Proof. We have $\widehat{G_{in}} = \bigcup_{i=1}^{k} (\widehat{G_i} \cup C_i) \cup \widehat{G_{k+1}}$. Lemma 4.2 and our assumption on length$(\widehat{G_i})$ imply

$$\begin{aligned} \text{length}(\widehat{G_{in}}) &\le 2^{poly(1/\epsilon)} W_{in} \\ &\le 2^{poly(1/\epsilon)} \mathsf{OPT}(G_{in}) \end{aligned}$$

where the last inequality uses Lemma 4.1.

Let L be an optimal solution to the original graph G_{in}. Define $L_i = (L \cap E(G[S_i])) \cup C_i$. Clearly L_i is a feasible solution for G_i.

Let $L_{k+1} = L \cap E(G_{k+1})$. Clearly L_{k+1} is a feasible solution for G_{k+1}.

Lemma 4.2 implies

$$(4.6) \qquad \sum_i \text{length}(L_i) < \text{length}(L) + \epsilon W_{in}$$

For $i = 1, \ldots, k+1$, let L'_i be a solution for G_i such that $L'_i \subset \widehat{G_i}$ and $\text{length}(L'_i) \leq \mathsf{OPT}(G_i) + c\epsilon\, W_i$.

Then $\bigcup_i (L'_i \cup C_i)$ is a solution for G_{in}, is a subset of $\widehat{G_{in}}$, and has length

$$\begin{aligned}
&\sum_i (\text{length}(L'_i) + \text{length}(C_i)) \\
&\quad \leq \sum_i (\text{length}(L_i) + c\epsilon\, W_i + \text{length}(C_i)) \\
&\quad \leq \text{length}(L) + c\epsilon W_{in} + 2\sum_i \text{length}(C_i) \\
&\qquad \text{by definition of } L_i \\
&\quad \leq \mathsf{OPT}(G_{in}) + c\epsilon\, 2\mathsf{OPT}(G_{in}) + 4\epsilon\, \mathsf{OPT}(G_{in}) \\
&\qquad \text{by Lemmas 4.2 and 4.1} \\
&\quad \leq (1 + (4+c)\epsilon)\mathsf{OPT}(G_{in})
\end{aligned}$$

which shows that it is a near-optimal solution. □

4.4 Interpretation of cuts $\delta_{G_0}(S)$ in the dual Recall that the edges of a simple cut $\delta_{G_0}(S)$ in the planar primal form a simple cycle in the planar gual G_0^*. Interpreting v_0 as the infinite face of G_0^*, the vertices in S are exactly the faces of G_0^* enclosed by this simple cycle.

This interpretation enables us to efficiently implement Line 4 of MainSpanner. Use the transfer-function technique of Section 3.2 to assign weights to the darts of G_0^* so that the weight of a clockwise simple cycle equals the weight of the enclosed faces. Define the length of a dart to be the length of its edge. For each dart d, assign cost $\text{cost}(d) \leftarrow \text{length}(d) - \epsilon\, \text{weight}(d)$. With this assignment, the cost of a clockwise cycle is negative iff the corresponding simple cut has length-weight ratio less than ϵ. To carry out Line 4, the algorithm therefore needs to find a minimally enclosing negative-cost cycle. This can be computed iteratively with the help of a subroutine for testing for the existence of a negative-cost cycle.

Lemma 4.3 in terms of dual cycles is:

Lemma 4.5. *For $i = 1, \ldots, k+1$, interpreting $v_{i,\infty}$ as the infinite face of the planar dual G_i^* of G_i, for any simple cycle C that is not the boundary of the infinite face, the weight enclosed by C is at most ϵ^{-1}length(C).*

5 Building a skeleton

We recommend that the reader review Figure 1 to recall the structure of a solution in the dual, because the algorithm Skeleton (G, v_∞) and its properties address primarily not G but its planar dual G^*. In G^*, v_∞ is considered the infinite face. The terminals are faces as well, so we refer to them as *terminal faces.* We interpret a subset L of edges as a subgraph of the dual G^*; e.g. when we discuss connected components of L, we mean

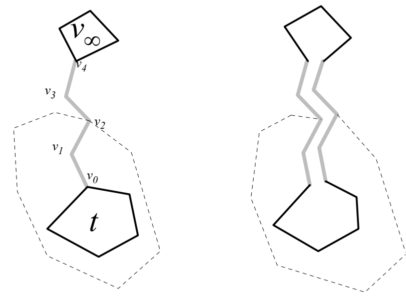

Figure 2: On the left is shown part of G^*. Shown in thick gray is a shortest path from a vertex of the face t to a vertex of the face v_∞. The dashed edges form a shortest cycle that encloses t and crosses the path at v_2. On the right is shown the graph G' obtained by cutting along the shortest path, duplicating its edges and vertices. This merges the faces t and v_∞. The cycle enclosing t is now a shortest path between the two copies of v_2.

connected components of the corresponding subgraph of G^*.

We present the algorithm in Section 5.1 and we present the correctness properties in Section 5.2.

5.1 The *skeleton* algorithm Refer to the pseudocode on the next page.

Step 1: Cycles. For each terminal t, the algorithm includes in the skeleton some cycles enclosing t, selected as follows. Find a shortest path $P = v_0 v_1 \cdots v_k$ connecting an arbitrary vertex on the face t to an arbitrary vertex on the infinite face v_∞. For each vertex v_i of P, we find a shortest cycle crossing P only at v_i, and include some of those cycles in the skeleton.

In order to find these cycles, cut the planar embedded dual graph G^* along P by duplicating the edges and vertices of P. Let G'^* be the result. For each vertex v_i of P, find a shortest path P_i in G' between the two copies of v_i. The edges of P_i form a cycle C_i in G that crosses P exactly one, and encloses t. The cycles can be chosen so that each (nonstrictly) encloses all the previous cycles.

To decide which of those cycles to add to the skeleton, for each $\ell = 0, 1, 2, \ldots, 3\epsilon^{-2}$, the algorithm considers the integers i for which $\text{length}(C_i) \leq \epsilon\,\ell\text{weight}(t)$. Let these integers be $i_1 < i_2 < \cdots < i_q$. The algorithm includes cycle C_{i_j} in the skeleton if $(w_{i_{j-1}}, w_{i_j}]$ contains an integer multiple of $\text{weight}(t)\epsilon$ or $w_{i_{j+1}} - w_{i_j} > \text{weight}(t)\epsilon$.

Step 2: Ears. Let H_1 be the result of Step 1. H_1 has the property that every connected component is 2-edge-connected. Step 2 preserves this property. An *ear* E of G with respect to H is a path in G that starts and ends on a connected component K of H and that does

Algorithm 3 SKELETON (G, v_∞)

Input: graph G and vertex v_∞ such that G is ϵ-short-cut-free with respect to v_∞
Output: skeleton H, a subset of the edges of G, with each edge of H marked as either a blob edge *or a* cluster edge*.*

Step 1 (cycles):

initialize $H_1 := \emptyset$
for each terminal face t in the dual G^* **do**
  $v_0 v_1 \ldots v_k \leftarrow$ shortest path in G^* from a vertex v_0 on face t to a vertex v_k on face v_∞.
  for $i = 0, \ldots, k$ **do**
    $C_i \leftarrow$ shortest cycle in G^* that encloses t and crosses $v_0 v_1 \ldots v_k$ at v_i, chosen so that C_i encloses C_{i-1} and $C_i \neq$ boundary of v_∞
    $w_i \leftarrow$ weight enclosed by C_i
  for $\ell = 0, 1, \ldots, \epsilon^{-2}$ **do**
    let $i_1 < i_2 < \cdots < i_q$ be the integers i for which $\text{length}(C_i) \leq \epsilon\, \ell\, \text{weight}(t)$
    for each j such that $(w_{i_{j-1}}, w_{i_j}]$ contains an integer multiple of $\epsilon\, \text{weight}(t)$ or $w_{i_{j+1}} - w_{i_j} > \epsilon\, \text{weight}(t)$ **do**
      $H_1 \leftarrow H_1 \cup C_{i_j}$

Step 2 (ears):

$H_2 := H_1$
while there is a component K of H_2 and an ear $\mathcal{E}$ of K such that
  $\mathcal{E}$ separates terminals t, t' and $\text{length}(\mathcal{E}) \leq \epsilon^{-3} \min\{\text{weight}(t), \text{weight}(t')\}$, or
  $\mathcal{E}$ separates terminal t from v_∞, and $\text{length}(\mathcal{E}) \leq \epsilon^{-3}\text{weight}(t)$
add $\mathcal{E}$ to H_2

Step 3 (clustering):

let $G^{*\prime} := G^*/H_2$, obtained from G^* by contracting the edges of H_2
comment: each vertex v of $G^{*\prime}$ is the result of merging the vertices of some connected component of H_2 (possibly a one-vertex component)
for each vertex v of $G^{*\prime}$ **do**
  $\phi(v) \leftarrow \epsilon^{-2}\text{length}(\text{the connected component merged to form } v)$
  unless this connected component contains the boundary of the face v_∞, in which case $\phi(v) \leftarrow \epsilon^{-2}\text{length}(G^*)$
$H_3' \leftarrow$ PC-CLUSTERING$(G^{*\prime}, \phi)$
$H_3 := H_2 \cup H_3'$, where the edges of H_3' are viewed as the corresponding edges of G^*.
return H_3

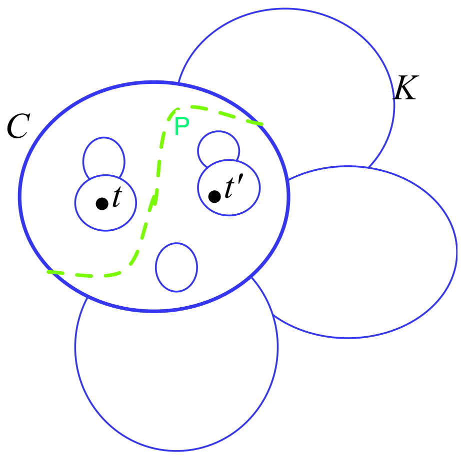

Figure 3: The dashed green path P is an ear that separates t from t'. Note that t, t' are already separated but not by the connected component K.

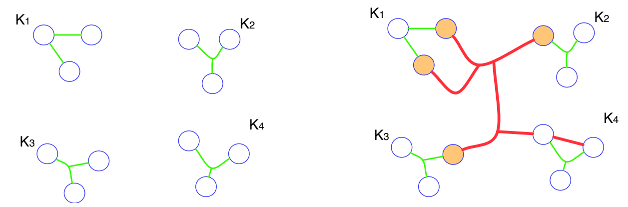

Figure 4: The figure on the left shows the four clusters found by PC-Clustering. The circles represent blobs (resulting from the *cycles* and *ears* steps). The thin green edges connecting the blobs into clusters were added to the skeleton during the *cluster* step. The figure on the right illustrates Property C of Lemma 5.4. The thick red lines form a connected component $\hat{K}$ of a near-feasible near-optimal solution L. The circles that are filled in are designated as *special* blobs. Note that the only nonspecial blobs connected by $\hat{K}$ belong to the same cluster.

not cross H (though it can share edges with H).[3] We say an ear E *separates* a pair of faces f_1, f_2 of G if the faces are in the same face of K but not of $K \cup E$. Refer to Figure 3 for an illustration.

The algorithm repeatedly adds ears whose lengths are small compared to the weights of terminals they separate[4]. The ear step also tries to separate terminals from the infinite face v_∞. The shortest ear separating a given pair of terminals can be found using a simple variant of the transfer-function technique reviewed in Section 3.2.

Step 3: Clustering. H_2 is defined to be H_1 together with the ears added in Step 2. A connected component of H_2 is called a *blob*. Note that each connected component is 2-edge-connected.

A blob that contains the boundary of the infinite face is called the *outer* blob, if it exists. Let $G^{*\prime} := G^*/H_2$ be the graph obtained from G by contracting the edges of H_2 (ignoring the planar embedding). The algorithm assigns a potential $\phi(v)$ to each vertex of $G^{*\prime}$ Each vertex v of $G^{*\prime}$ is either a vertex of G^* (in which case $\phi(v) = 0$ or is the result of contracting a blob B. (in which case $\phi(v) = \epsilon^{-2}\text{length}(B)$ or, if B contains the boundary of v_∞, $\phi(v) = \epsilon^{-2}\text{length}(G^*)$..

The algorithm runs the PC-Clustering algorithm on $(G^{*\prime}, \phi)$ to obtain a forest H_3' connecting some of the blobs to each other. The skeleton is then defined as $H_3 := H_2 \cup H_3'$, where the edges of H_3' are viewed as the corresponding edges of G.

A connected component of H_3 after completion of the clustering step is called a *cluster*. The edges selected in this step are called *cluster edges*. We say a blob is *of* a cluster or *belongs* to a cluster if the blob is a subset of the cluster.

5.2 Structure of the skeleton and a near-optimum solution

Definition of $C(L,t)$ and of $K(L,t)$ Let L be a set of edges of G (a feasible multiway cut for G and some subset of terminals). We consider L as a subset of the edges of the planar dual G^*. Let t be a terminal face of G^*. Among the simple cycles in G^* that consist of edges of L, we use $C(L,t)$ to denote the minimally enclosing cycle that encloses t (if such exists, see Figure 1). $C(L,t)$ is called the *cycle of* t in L. Among the connected components of L considered as a subgraph of G^*, the component that contains $C(L,t)$ is denoted by $K(L,t)$.

Definition of W, of R-feasible, and of near-optimal Let W denote the sum of weights of vertices of G. For a subset T' of T, we say that a set of edges of G is a *T'-feasible solution* if the edges form a feasible multiway cut for all terminals in $T' \cap V(G)$. We say it is a *near-optimal T'-feasible solution* if in addition weight$(T - T') \leq c\epsilon W$ and the set of edges has length at most $(1 + c\epsilon)OPT + c\epsilon W$ where OPT refers to the optimal length of a true solution (and, again, c is a constant).

Lemma 5.1. *There exists a subset R of terminals and a near-optimal $(T - R)$-feasible solution L such that,*

> **Property A:** *for each terminal $t \notin R$, at least one cycle added for t in the* cycles *step intersects $K(L,t)$.*

Proof. We give an algorithm that, given an optimal solution L_0, computes sets L_1 and R_1 such that L_1

[3] This definition of ears differs from the traditional definition.

[4] Beware that those terminals might already be separated, but the ear is attached to a particular connected component, and "separates" means that those terminals were not separated by that component until the ear was added.

is a near-optimal $(T - R_1)$-feasible solution. Initially $L_1 \leftarrow L_0$ and $R_1 \leftarrow \emptyset$. For each teminal t, we process t as follows. Consider the cycle $C(L_1, t)$ minimally enclosing t in L_1. Denote this cycle by C^t.

Assume C^t does not intersect any of the cycles added to the skeleton when considering t. In this case, we will modify L_1. Let ℓ^* be the integer such that $\epsilon^{-1}\text{length}(C^t)/\text{weight}(t) \in (\ell^* - 1, \ell^*]$. If $\ell^* > \epsilon^{-2}$, we add t to R_1. Else, let v_{i^*} denote a vertex at which C^t crosses the shortest path $v_0 v_1 \ldots v_k$ in the *cycles* step. It follows that $\text{length}(C_{i^*}) \leq \epsilon^{-1}\text{weight}(t)$, so during $i^* = i_q$ for some q in Line 8 of the *cycles* step during iteration ℓ^*. Let p and r be the integers such that

1. $p < q < r$,
2. C_{i_p} and C_{i_r} were added to H_1 in Line 10, and
3. for every integer s in the interval (p, r), C_{i_s} was not added.

We modify L_1 by adding all the edges of C_{i_r} and deleting all the edges of C^t and deleting any connected components that are strictly enclosed by C_{i_k} and not enclosed by C^t. Since $\text{length}(C_{i_k}) \leq \epsilon^{-1}\ell^*\,\text{weight}(t)$ and $\text{length}(C^t) \geq (\ell^* - 1)\epsilon\,\text{weight}(t)$, the increase in $\text{length}(L_1)$ is at most $\epsilon\,\text{weight}(t)$.

After the modification, $K(L_1, t)$ includes C_{i_r} so the modification achieves Property A for t. We show below that it preserves it for terminals that have already been processed.

To preserve the $(T - R_1)$-feasibility of L_1, we add to R_1 all the terminals enclosed by C_{i_r} but not by C^t. The weight of these terminals is $w_{i_r} - w_{i_p}$. C_{i_r} encloses v_i but by assumption does not intersect C^t so must enclose C^t. Similarly, C_{i_p} must be enclosed by C^t. Hence the weight of all terminals enclosed by C_{i_r} but not by C^t is at most $w_{i_r} - w_{i_p}$. By Property 3, $(w_{i_p}, w_{i_{r-1}}]$ contains no integer multiple of $\epsilon\,\text{weight}(t)$, and $w_{i_r} - w_{i_{r-1}} < \epsilon\,\text{weight}(t)$, so $w_{i_r} - w_{i_p} < 2\epsilon\,\text{weight}(t)$. Thus the increase in $\text{weight}(R_1)$ is at most $2\epsilon\,\text{weight}(t)$.

We must show that Property A continues to hold for each previously processed terminal t'. If t' is enclosed by C_{i_r} but not by C^t then t' is added to R_1 so the property holds trivially. Suppose t' is enclosed by C^t. Then before the modification $K(L_1, t')$ is strictly enclosed by C^t, so the modification does not change $K(L_1, t')$. Suppose t' is not enclosed by C_{i_r}. Then $K(L_1$ before modification$, t)$ cannot be strictly enclosed by C_{i_r} so none of it is removed by the modification, and the addition of C_{i_r} to L_1 only adds to $K(L_1, t)$.

Summing over all iterations of the outer for-loop, the total increase in $\text{length}(L_1)$ is at most $\epsilon \sum_t \text{weight}(t)$ and $\text{weight}(R_1)$ is at most $2\epsilon \sum_t \text{weight}(t)$.

□

We use $C(t)$ to refer to the cycle whose existence is asserted in Property A.

LEMMA 5.2. *There exists a subset R of a terminals and a near-optimal $(T - R)$-feasible solution L that satisfies Property A and also*

> **Property B:** *Let B be a blob and let F be a finite face of B. Then there is at most one terminal $t \notin R$ enclosed by F such that there is no blob B' enclosed by F that intersects $K(L, t)$.*

Proof. Let R_1 be the set of terminals defined in Lemma 5.1 and let L_1 be the near-optimal $(T - R_1)$-feasible solution of that lemma. We define a set R_2 of terminals by the the following algorithm. For each blob B and each face F of B in turn, let T_F denote the set of terminals t enclosed by F, not in R, and such that there is no blob B' enclosed by F that intersects $K(L_1, t)$. If T_F has size 2 or more, add all the terminals of T_F to R_2. At the end, we let $R = R_1 \cup R_2$.

By construction, the second property holds. It remains to bound the weight of R_2. Consider a blob B and a face F of B such that T_F has size at least 2.

Let t, t' be two distinct terminals of T_F. Since L_1 is feasible, it separates t from t', and up to exchanging the role of t and t' we can assume that $C(L_1, t)$ separates t from t'. Then some edges of $C(L_1, t)$ must be inside F. But $C(L_1, t)$ also intersects $C(t)$, which encloses F, so $C(L_1, t)$ intersects the boundary of F, forming an ear $\mathcal{E}$ that separates t from t'. Since this ear was not added in the *ears* step,

$$\min(\text{weight}(t)), \text{weight}(t')) < \epsilon^3 \text{length}(\mathcal{E}).$$

Let $t_{\min}$ be the terminal in T_F of minimum weight. The cycle $C(t_{\min})$ encloses F and therefore encloses all terminals in T_F, so $\text{weight}(T_F) \leq$ weight enclosed by $C(t_{\min})$. By Lemma 4.5 (and since $C(t_{\min})$ is not the boundary of the infinite face), weight enclosed by $(C(t_{\min})) \leq \epsilon^{-1}\text{length}(C(t_{\min}))$. By definition of the cycle step, $\text{length}(C(t_{\min})) \leq \epsilon^{-1}\text{weight}(t_{\min})$. Thus

$$\text{weight}(T_F) \leq \epsilon^{-2}\text{weight}(t_{\min}).$$

Combining, $\text{weight}(T_F) \leq \epsilon\,\text{length}(\mathcal{E})$. By definition of T_F, no blob in F encloses any edge of $\mathcal{E}$. Thus as we sum over all B and F, the ears are all disjoint sets of edges, and we obtain $\text{weight}(R_2) \leq \epsilon\,\text{length}(L_1)$. □

The algorithm does not know L or R, so cannot uniquely identify the one terminal t mentioned in Property B. However, the following lemma gives us a technique to identify a bounded number of possible candidates.

LEMMA 5.3. *Let Y be a subgraph. Let $\tilde{T}_Y$ denote the set of terminals t enclosed by Y such that some cycle C chosen for t in the* cycle *step encloses Y. Then $|\tilde{T}_Y| \leq \epsilon^{-2}$.*

Proof. Let $t_{\min}$ be the terminal in $\tilde{T}_Y$ of minimum weight. Then

$$|\tilde{T}_Y| \leq \text{weight}(\tilde{T}_Y)/\text{weight}(t_{\min}).$$

Let C be the cycle chosen for $t_{\min}$ that encloses Y. C therefore encloses all terminals in $\tilde{T}_Y$, so weight($\tilde{T}_Y$) is at most the weight enclosed by C. By Lemma 4.5 (and since C is not the boundary of the infinite face), weight$(C) \leq \epsilon^{-1}$length(C). By definition of the cycle step, length$(C) \leq \epsilon^{-1}$weight$(t_{\min})$. Thus

$$\text{weight}(\tilde{T}_Y) \leq \epsilon^{-2}\text{weight}(t_{\min}).$$

□

LEMMA 5.4. *There exists a subset $\hat{R}$ of terminals and a near-optimal $(T - \hat{R})$-feasible solution $\hat{L}$ that satisfies Properties A and B and also*

> **Property C:** *There is a set of* special *blobs such that (i) if $\hat{L}$ connects two blobs in different clusters then at least one is special, (ii) no edge of $\hat{L}$ is properly enclosed by a special blob, and (iii) R contains every terminal enclosed by a special blob*

Proof. Let R_2 be the set of terminals defined in Lemma 5.2, and let L_2 be the near-optimal $(T - R_2)$-feasible solution defined in that lemma. Let L_2' be obtained from L_2 by contracting each blob B of H_2. The *cluster* step runs PC-CLUSTERING on $(G^{*\prime}, \phi)$, obtaining the subgraph H_3'. We apply part 2 of Theorem 3.1 to the subgraph L_2' of G' This part asserts the existence of a set Q of vertices of $G^{*\prime}$, which correspond to blobs in G^*.

We say a blob B is *outer* if it contains all the edges of the infinite face v_∞ of G^*. We designate a blob B as *special* if the corresponding vertex of $G^{*\prime}$ belongs to Q. Part 2a of Theorem 3.1 implies that no outer blob is special. We define R_3 as the set of all terminals that in G^* are enclosed by special blobs, and we set $\hat{R} = R_2 \cup R_3$. Finally, we define $\hat{L}$ to include edges in L_2 not strictly enclosed by special blobs, and also all the edges on the outer boundaries of special blobs. This ensures that $\hat{L}$ is $(T - \hat{R})$-feasible, and implies Parts (ii) and (iii) of the present lemma. Part 2b of Theorem 3.1 implies that $\hat{L}$ satisfies part (i).

Part 2a of Theorem 3.1 implies that

$$\begin{aligned} &\text{length}(L_3) - \text{length}(L_2) \\ &\quad\leq \sum_{B \text{ special}} \text{length}(B) \\ &\quad\leq \epsilon^2 \sum_{v \in Q} \phi(v) \leq \epsilon^2 \text{length}(L_2) \end{aligned}$$

since no outer blob is special.

Additionally, by Lemma 4.5 and since an outer blob is not special,

$$\text{weight}(R_3) \leq \epsilon^{-1} \sum_{B \text{ special}} \text{length}(\partial B) \leq \epsilon \, \text{length}(L_2).$$

Finally, it is easy to verify that Property A of Lemma 5.1 still holds for $\hat{L}$, and Property B of Lemma 5.2 holds as a matter of course. □

The clusters and enclosure relation between clusters naturally induce a nesting forest of clusters. Let $\texttt{Enclosed}(K)$ denote the terminals enclosed by cluster K.

Definition of *structured solution* A *structured solution* with respect to a subset R of terminals is a multiset of edges S that can be partitioned into sets, $S = \cup_{K \text{ cluster}} S_K$, in such a way that for every K, $\cup\{S_{K'} : K'$ enclosed by $K\}$ is a feasible multiway cut for $\texttt{Enclosed}(K) - R$. Moreover, if no cluster properly encloses K, then in addition $\cup\{S_{K'} : K'$ enclosed by $K\}$ also separates $\texttt{Enclosed}(K) - R$ from t_∞.

For each cluster K of the skeleton H, define

$$\begin{aligned} \hat{L}_K = \bigcup\{\hat{K} :\ &\hat{K} \text{ a connected component of } \hat{L} \\ &\text{that intersects some non-special blob of } K\}. \end{aligned}$$

THEOREM 5.1. *$(\hat{L}_K)_K$ is a partition of $\hat{L}$ that is a structured solution with respect to $\hat{R}$. Moreover, for each cluster K of the skeleton H, we have:*

1. *Every edge of $\hat{L}_K$ is reachable from cluster K without crossing into a blob of any other cluster. (Equivalently: $\hat{L}_K$ is in a single face of $H - K$.)*

2. *If some terminal t in $\texttt{Enclosed}(K) - \hat{R}$ is not separated from t_∞ by*

   $$\bigcup\{\hat{L}_{K'} \ :\ K' \textit{ properly enclosed by } K\}$$

   then $C(t)$ encloses the face F of K that encloses t.

3. *Let Mandate$(\hat{L}_K)$ be the set of pairs $\{t, t'\}$ of terminals in $\{t_\infty\} \cup \texttt{Enclosed}(K) - \hat{R}$ that are separated by $\hat{L}_K$ and that are not already separated by*

   $$\bigcup\{\hat{L}_{K'} \ :\ K' \textit{ properly enclosed by } K\}$$

   Then $\{t : t$ appears in Mandate$(\hat{L}_K)\}$ has at most one terminal per face of K.

Proof. To prove that we have a structured solution, consider two terminals t_1, t_2 that are not in $\hat{R}$ and are enclosed by some cluster K. Since $\hat{L}$ is feasible for terminals not in $\hat{R}$, up to exchanging the roles of t_1 and of t_2 we can assume that $K_{\hat{L}}(t_1)$ separates t_1 from t_2. By Property A (Lemma 5.1), $K(\hat{L}, t_1)$ intersects a cycle $C(t)$ that is part of a blob of some cluster K_1 of the skeleton H (a non-special blob, since $t_1 \notin \hat{R}$ and all terminals enclosed by special blobs are in $\hat{R}$ by Property C of Lemma 5.4). If cluster K_1 is enclosed by K then $K(\hat{L}, t_1)$ is in $\bigcup\{\hat{L}_{K'} : K' \text{ properly enclosed by } K\}$ and so t_1 and t_2 are separated by that union, as desired. If K_1 encloses K, then $K(\hat{L}, t_1)$ simultaneously intersects a non-special blob of K_1 but also separates two terminals enclosed by K, hence must also intersect a non-special blob of K: that contradicts Lemma 5.4.

In terms of separation from t_∞, we know that $\hat{L}$ is feasible, so every $t \notin \hat{R}$ must be separated from t_∞ by some $K(\hat{L}, t)$, that (by Property A) intersects the non-special blob of $C(t)$, so if K denotes the cluster of that blob, then K encloses t and $K(\hat{L}, t)$ is in $\hat{L}_K$, hence $\hat{L}_K$ separates t from t_∞, as desired. This proves that $\hat{L}$ is a structured solution with respect to $\hat{R}$.

To prove Property 1, consider a connected component $\hat{K}$ of $\hat{L}_K$. By definition it intersects a non-special blob of K. By Lemma 5.4 it does not intersect any other non-special blob and does not enter into any L-special blob. This implies the desired property.

To prove Property 2, let t be in $\texttt{Enclosed}(K) - \hat{R}$ and not separated from t_∞ by $\cup\{\hat{L}_{K'} : K'$ properly enclosed by $K\}$. In other words, $K(\hat{L}, t)$ is not in $\hat{L}_{K'}$ for K' enclosed by K. By Lemma 5.1, $K(\hat{L}, t)$ intersects cycle $C(t)$ of the skeleton. This cycle is part of a blob of a cluster, call it K_1, and by definition of $\hat{L}_{K_1}$, $K(\hat{L}, t)$ is in $\hat{L}_{K_1}$. So K_1 is not any K' enclosed in K, and so $C(t)$ must enclose the face F of K that encloses t.

To prove Property 3, consider a cluster K and a face F of K and study the terminals pairs of Mandate($\hat{L}_K$) that involve at least one terminals in F. By Lemma 5.2, every terminal t' in F except at most 1 (call that special terminal t_F) is in a blob B' (part of some K') enclosed in F and that intersects $K(\hat{L}, t')$, so $K(\hat{L}, t')$ is in $\hat{L}_{K'}$, and it encloses t'; moreover by Lemma 5.4 $K(\hat{L}, t')$ cannot intersect any blob of K, so it has to stay enclosed by F, so $\hat{L}_{K'}$ separates t' from everyone outside F, and so Mandate($\hat{L}_K$) does not contain any pair (t', t") with t" outside F. Any two terminals $t'_1, t'_2 \neq t_F$ that are in F are separated by at least one of $K(\hat{L}, t'_1)$ and $K(\hat{L}, t'_2)$, hence separated by $\hat{L}_{K'_1} \cup \hat{L}_{K'_2}$, so that pair also cannot appear in Mandate($\hat{L}_K$). For t' and t_F, since we already have $K(\hat{L}, t')$ in $\hat{L}_{K'}$, the only way in which t_F and t' could be not separated would be if $K(\hat{L}, t')$ enclosed both t' and t_F, and then $K(\hat{L}, t_F)$ would have to be enclosed in $K(\hat{L}, t')$. But that would contradict the fact that $K(\hat{L}, t')$ is enclosed in F whereas $K(\hat{L}, t_F)$ must intersect a blob by Lemma 5.1, and that has to be F or outside F. This proves the theorem. $\square$

THEOREM 5.2. *The length of the output H of the skeleton algorithm is $O(\epsilon^{-8})W$.*

Proof. First we analyze H_1, the result of the *cycle* step. Consider the iteration for a terminal t. A cycle C_i cannot enclose more than length$(C_i)/\epsilon$ weight by Lemma 4.5. Since length$(C_i) \leq \epsilon\, \ell$weight$(t)$ and $\ell \leq \epsilon^{-2}$, the enclosed weight is at most ϵ^{-2} weight(t). For each condition in Line 9, the enclosed weight changes by at least ϵ weight(t) from one selected cycle to the next, so there are at most $2\epsilon^{-3}$ cycles added for each value of t and of ℓ. There are ϵ^{-2} values of ℓ, and each cycle has length at most ϵ^{-1}weight(t), so the total length added for terminal t is at most $2\epsilon^6$weight(t). Summing over t yields

$$\text{length}(H_1) \leq 2\epsilon^{-6}W.$$

Now we analyze H_2. Just for for this part of the proof, to avoid the special case of $\mathcal{E}$ separating a terminal from v_∞, we imagine that v_∞ is a terminal and that it has the largest weight. Whenever the algorithm adds an ear $\mathcal{E}$ to a component K, consider the terminals t, t' separated by $\mathcal{E}$ that have maximum weight, and charge the length of $\mathcal{E}$ to whichever of the two terminals has minimum weight, resolving ties in a consistent manner, assuming for example, up to an infinitesimal perturbation, that all weights are distinct.

We claim that each terminal gets charged at most once. To see this, assume, for a contradiction, that t_0 gets charged, first by an ear $\mathcal{E}$ (in face F of component K) separating t_0 from t_1, and then later by an ear $\mathcal{E}'$ (in face F' of component K') separating t_0 from t_2. By the definition of charging, weight$(t_0) <$ weight(t_1) and weight$(t_0) <$ weight(t_2). Ear $\mathcal{E}$ splits F into two faces, F_0 and F_1, containing t_0 and t_1 respectively. Ear $\mathcal{E}'$ splits F' into two faces, F'_0 and F'_1, containing t_0 and t_2 respectively. Face F' either is enclosed in F_0 (possibly with equality), or encloses K. In the first case, t_2 is in face F_0, contradicting the maximality of weight(t_0) among terminals in F_0; in the second case, t_1 is in face F'_0, contradicting the maximality of weight(t_0) among terminals in F'_0. Thus the claim holds. Finally, since v_∞ has the largest weight, it never gets charged, and so the total length of the ears added is at most $\epsilon^{-3} \sum_{t \neq \hat{v}}$ weight$(t) = \epsilon^{-3}W$, so

$$\text{length}(H_2) \leq \text{length}(H_1) + \epsilon^{-3}W.$$

Finally we analyze H_3. By construction length$(H_3) \leq$ length(H_2) + length(H'_3). Using Theorem 3.1 and the definition of potential:

$$\begin{aligned}\text{length}(H_3') &\leq 2\sum_v \phi(v) \\ &\leq 4\sum \epsilon^{-2}\{\text{length}(K) \;:\; K \text{ a blob}\} \\ &= 4\epsilon^{-2}\text{length}(H_2)\end{aligned}$$

where the second inequality follows since each blob B is counted once for its own potential an d possibly once for the outer blob.

Combining,

$$\text{length}(H_3) \leq (4\epsilon^{-2}+1)(2\epsilon^{-6}W + \epsilon^{-3}W) \leq c\epsilon^{-8}W.$$

for a constant c. □

6 Building a spanner

Like the skeleton algorithm, the spanner algorithm operates in the planar dual.

6.1 The spanner algorithm Algorithm SPANNER outlines the spanner algorithm. We use the brick decomposition algorithm from Section 3.4. For each connected component K of the skeleton, the spanner algorithm constructs a brick decomposition. Let M_K be the mortar graph. By Inequality 3.4, length(M_K) is $O(\epsilon^{-1})$length(K). Therefore, by Theorem 5.2, summing over all clusters K, the total length of all the brick decompositions is $O(\epsilon^{-c_2})W$ where c_2 is a constant. By Inequality 3.5, we can choose a constant c_1 so that, when we set $\kappa = \epsilon^{-c_1}$ for every brick decomposition, we ensure that the east and west boundaries have total length $O(\epsilon)W$.

For a constant c_3 to be determined at the end of this section, we set $\theta = \epsilon^{-c_3}$. For each brick B, the algorithm selects θ portals along the boundary ∂B of B. The simple greedy selection rule is described in [7]. It ensures that, for any vertex x on ∂B, there is a portal y such that the x-to-y subpath of ∂B has length at most length$(\partial B)/\theta$.

The portals divide ∂B into a set $\mathcal{P}_b$ of θ subpaths. A *configuration* is a pair (π, S) where π is a partition of the subpaths $\mathcal{P}_b$ and S either is one of the parts of π or is $\emptyset$.

The following definitions are illustrated by Figures 5 and 6. A set $\mathcal{E}$ of edges of brick B is *consistent* with partition π if the following condition holds:

> if two edges e_1, e_2 of ∂B that are in subpaths in different parts of π, then they are separated by $\mathcal{E}$.

Given a terminal t, $\mathcal{E}$ is consistent with (π, S) if in addition the following condition holds:

> if $e \in \partial B$ is in a subpath not in part S of π, then e and t are separated by $\mathcal{E}$.

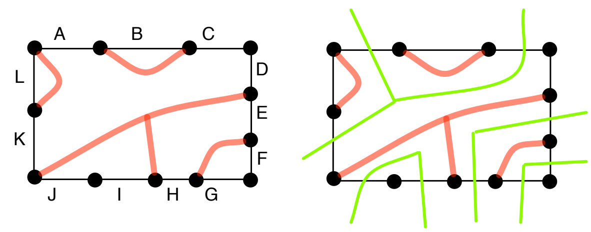

Figure 5: The figure on the left represents a brick b. The big dots are the portals, and the thick red lines consist of edges in E. The subpaths of ∂B are labeled by the letters A through L. The corresponding partition π is $\{\{A,C,D,K\},\{B\},\{E,H\},\{F,G\},\{I,J\},\{L\}\}$. The figure on the righ illustrates the definition of consistency. The thinner green lines represent paths in the dual. (The outgoing edges all lead to the vertex representing the infinite face but that vertex is not shown.)

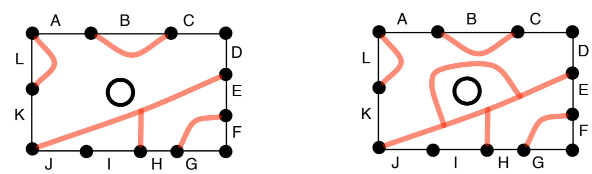

Figure 6: A terminal is indicated by the circle. The figure on the left is consistent with the pair $(\pi, \{A,C,D,K\})$ where π is the partition of Figure 5. The second element of the pair is the part that corresponds to the region containing the terminal. The figure on the right is consistent with the pair $(\pi, \emptyset)$ since there is no way to get from an edge of ∂B to the terminal while avoiding E.

A *portal-respecting solution* is a multiway cut that, in the planar dual, only crosses brick boundaries at portals. We later show that, in a sense, it is possible to restrict our attention to portal-respecting solutions. The spanner algorithm therefore proceeds as follows. For each brick B, the algorithm identifies the set $\tilde{T}_B$ of terminals in B for which some cycle selected in the *cycle* step encloses B. By Lemma 5.3, $|\tilde{T}_B| \leq \epsilon^{-2}$. For each terminal $t \in \tilde{T}_B$, the algorithm enumerates the configurations of B and t. For each configuration, the algorithm finds an approximately minimum-length set $\mathcal{E}$ of edges consistent with the configuration, and includes $\mathcal{E}$ in the spanner. The spanner consists of all these edge-sets, together with the brick decomposition and the min-cuts of all terminals.

Given a configuration in the brick B and given a terminal t, SPANNER uses the following auxiliary algorithm to find an near-optimal set of edges that is consistent with that configuration, and SPANNER adds that set of edges to the spanner. Let p denote a shortest path from ∂B to t. Let p_{near} denote the part of p closest to ∂B, such that length(p_{near}) = min(length(∂B), length(p)), and let p_{far} denote the rest

Algorithm 4 SPANNER (G, v_∞, H)

Input: instance (G, v_∞) equipped with skeleton H (a subgraph of the planar dual G^*)
Output: spanner

include in the spanner the min-cuts for all terminals.
for each connected component K of the skeleton H,
  define a subgraph G_K of G as follows:
    retain an edge iff it is on a path that starts on K and that does not cross H.
  find a brick decomposition M of G_K with respect to K, with $\kappa = \epsilon^{-c_1}$
  include M in the spanner
  for each brick B,
    select $\theta = \epsilon^{-c_3}$ portals on ∂B
    let $\tilde{T}_B := \{t \in B :$ some cycle selected for t encloses $B\}$
    for each configuration of B,
      for each terminal $t \in \tilde{T}_B$ and for the no-terminal case,
        include in the spanner a near-optimal solution for that brick
        that is consistent with the configuration

of path p. The auxiliary algorithm places a collection of $\theta' = \epsilon^{-c_4}$ equidistant portals along p_{near}, thus partitioning p into subpaths $\mathcal{P}_{b,t}$. (The choice of c_4 will be made presently.)

We claim that there exists a near-optimal set of edges that only crosses p_{near} at portals. To see this, first observe that the solution has length at most length(∂B), since otherwise we could always replace it by ∂B; thus it can only cross p_{near}, not p_{far}. We modify the solution by adding detours through portals, to get a solution that only crosses p_{near} at portals. Theorem 3.2 ensures that we can restrict our attention to solutions with $O(\epsilon^{-2.5}\kappa)$ joining vertices (plus possibly one single cycle), so we can restrict our attention to solutions consisting of $O(\epsilon^{-2.5}\kappa)$ shortest paths. Each crosses p_{near} at most once, so the total cost of the detours is $O(\epsilon^{-2.5}\kappa)\text{length}(\partial B)/\theta'$. By Theorem 5.2, there is a choice of c_4 so that the total cost of the detours is $O(\epsilon)W$.

To find the best solution that crosses p_{near} at portals only, we make two copies $p^{(1)}$ and $p^{(2)}$ of p, duplicating every vertex except t, thus drawing a slit into b. For each possible extension of partition π into a partition of $\mathcal{P}_b \cup \mathcal{P}_{b,t}$, we solve the problem optimally via a dynamic program inside the brick (the details are omitted here since an analogous dynamic program has been described in [7, 13]). The runtime of the dynamic program is $2^{poly(1/\epsilon)} n_b \log n_b$. Among all solutions thus constructed, it only remains to pick the one of minimum cost.

Summing over all bricks and all connected components of the skeleton, the runtime to build a spanner given the skeleton is $2^{poly(1/\epsilon)} n^d$ for a constant d.

6.2 Structure of the spanner and a near-optimum solution

We transform the near-optimal solution $\hat{L}$ of Lemma 5.4 into a portal-respecting, yet almost as good, solution $\hat{L}''$. To do so, for each cluster K, the subset $\hat{L}_K$ of $\hat{L}$ as defined before Theorem 5.1, we construct a new subset $\hat{L}''_K$ that respects the mandates defined for $\hat{L}_K$.

The proof of the theorem relies on the facts that (1) each brick of the decomposition contains at most one terminal of Mandate$(\hat{L}_K)$; (2) each connected component of $\hat{L}_K$ is connected to K (and therefore to brick boundaries: i.e., there is no component floating unattached inside a brick); and (3) the skeleton has length $O(W)$. Given these, the subsection is an adaptation of [6].

To analyze the brick decomposition, we use Theorem 3.2 to prove the following theorem.

THEOREM 6.1. *For each cluster K, there exists a set of edges $\hat{L}''_K$ that*

1. *is portal-respecting,*
2. *has length at most $(1 + c\epsilon)$length$(\hat{L}_K) + \frac{O(\epsilon^{-2.5})\kappa}{\theta}$length$(M_K)$ for some constant c,*
3. *still satisfies Properties 1 and 2 of Theorem 5.1,*
4. *still intersects the non-special blobs intersected by $\hat{L}_K$, and*
5. *still separates every pair $\{t, t'\}$ in Mandate$(\hat{L}_K)$.*

Proof. We will show how to transform $\hat{L}_K$ into a solution $\hat{L}'_K$ that, for each brick B, crosses B's boundary only $O(\epsilon^{-2.5}\kappa)$ times. To build $\hat{L}''_K$ from $\hat{L}'_K$, we simply take, for each crossing, a detour to the nearest portal and back. A detour along the boundary of brick B

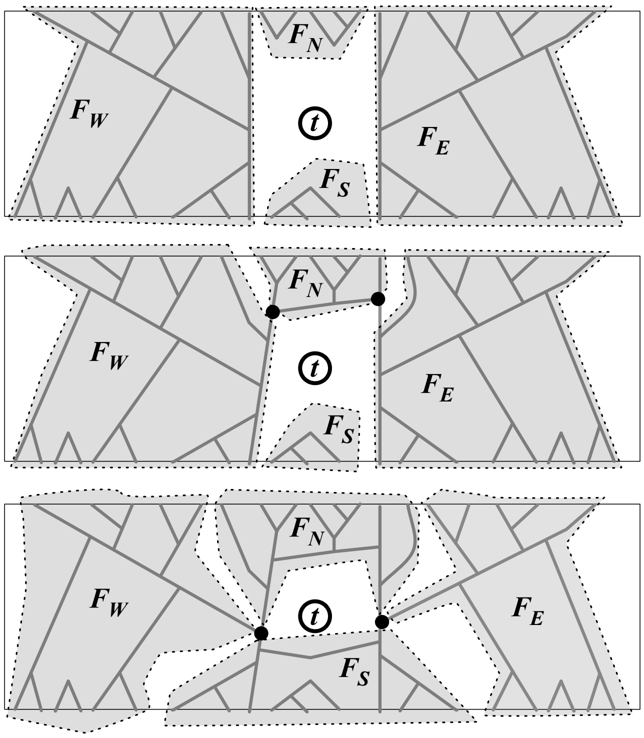

Figure 7: The three subcases in the proof of Theorem 6.1.

has length $\frac{1}{\theta}\text{length}(\partial B)$. Thus the total increase due to detours is $\frac{O(\epsilon^{-2.5}\kappa}{\theta}\text{length}(M_K)$

In reducing the number of crossings, we consider two cases. In the first case, there is no terminal of Mandate($\hat{L}_K$) inside the brick. Inside the brick, the near-optimal solution is a forest F. We apply Theorem 3.2 to F with $U = \emptyset$. Since the new forest F' preserves connectivity, it also preserves separation: if some interior path from some $x \in \partial B$ to some $y \in \partial B$ crosses F, then it also crosses F'. Thus, replacing F by F' preserves feasibility of our multiway cut solution.

In the second case, there is a single terminal t of Mandate($\hat{L}_K$) inside the brick. The proof uses the following definition. Given a forest F in the brick B, by N_F (resp. S_F) we denote the subpath of N (resp. S) spanning the vertices of $F \cap N$ (resp. $F \cap S$). Given a terminal t, we say that t is outside F if t is not enclosed by $F \cup N_F \cup S_F$. We say that F is *north* of t (resp. *south* of t) if t is outside F and $S_F = \emptyset$. We say that F is *east* of t (resp. *west* of t) if t is outside F and F separates t from E (resp. from W). Then we can decompose F into at most four forests, F_N, F_S, F_W, F_E, respectively North, South, West and East of t. More precisely, there are three subcases. See Figure 7.

In the first subcase, the cell of t intersects both N and S.

In the second subcase, the cell of t intersects just one side, S for example. Then there exist two vertices u_0 and u_1 in B such that u_0 is a leaf of F_W and of F_N, and u_1 is a leaf of F_E and of F_N.

In the third subcase, the cell of t intersects neither S nor N, and then there exist two vertices u_0 and u_1 in B such that u_0 is a leaf of F_W, F_N and F_S, and u_1 is a leaf of F_E, F_N and F_S.

It only remains to apply Theorem 3.2 to each of the four forests with $U \subseteq \{u_0, u_1\}$, defined appropriately for each forest. Thanks to the fourth property of Theorem 3.2, the mandates of our solution are preserved. □

Consider the set of edges put to the spanner when guessing configuration and terminals correctly. A new problem may arise: that collection still respects the mandates of each $\hat{L}_K$, but it is not necessarily a feasible solution overall. The following theorem patches the solution by identifying more terminals as being of negligible weight.

THEOREM 6.2. *There exist a set $\hat{R}'$ of weight at most $O(\epsilon)\,W$ such that $\hat{L}''$ is a $(T - \hat{R}')$-feasible solution.*

Proof. We start as in the proof of Lemma 5.4. Let $\tilde{L}$ be obtained from $\hat{L}''$ by contracting each blob B of H_2. The *cluster* step runs PC-clustering on $(G^{*\prime}, \phi)$, obtaining the graph H_3'. We apply Part 2 of Theorem 3.1 to the subgraph $\tilde{L}$ of G'.[5] This part asserts the existence of a set $\tilde{Q}$ of vertices of $G^{*\prime}$, which correspond to blobs in G^*. We designate a blob B as *special* if the corresponding vertex of $G^{*\prime}$ belongs to $\tilde{Q}$. We define R_1 as the set of all terminals that in G^* are enclosed by special blobs.

Part 2a of Theorem 3.1 and Lemma 4.5 imply that

$$\text{weight}(R_1) \leq 2\epsilon W$$

By Part 2b of Theorem 3.1, if two non-special blobs are connected by $\hat{L}''$, then they are in the same connected component of the skeleton.

We claim that every terminal pair of $T - (\hat{R} \cup R_1)$ is separated, except for a set R_2 of small weight. The proof has several cases.

First, consider the case in which one of the two terminals in the pair is t_∞. Let t_1 denote the other terminal. By Theorem 5.1, $\hat{L}$ is $(T - \hat{R})$-feasible, so t_1 and t_∞ are separated in $\hat{L}$, and we can consider a cluster K minimally enclosing t_1, separating t_1 from t_∞. By definition of mandates, $\{t_1, t_\infty\}$ is in Mandate(K), so Theorem 6.1, Part 5, ensures t_1, t_∞ are still separated in $\hat{L}''_K$ and hence in $\hat{L}''$.

Second, consider the case in which neither of the two terminals t_1, t_2 is t_∞. Let B_1 be the blob enclosing t_1 in the cluster K_1 of the skeleton that (from Lemma 5.1)

[5] Note that we do not need to run PC-CLUSTERING again.

has a cycle intersecting the component of $S = \hat{L}$ minimally enclosing t_1. We define B_2 and K_2 similarly.

Consider the subcase where $K_1 = K_2$. Then $\{t_1, t_2\}$ is in Mandate(K_1) and so they are separated in $\hat{L}''$.

Consider the subcase where $K_1 \neq K_2$ and K_1 and K_2 do not enclose each other. Since $\hat{L}_{K_1}$ minimally separates t_1 from t_∞, we have $\{t_1, t_\infty\} \in$ Mandate$(\hat{L}_{K_1})$. Hence, $\hat{L}''_{K_1}$ also separates t_1 from t_∞ (due to Theorem 6.1, Part 5). By Theorem 6.1 Part 4, $\hat{L}''_{K_1}$ still intersects B_1. Similarly $\hat{L}''_{K_2}$ separates t_2 from t_∞ and still intersects B_2. Assume, for a contradiction, that t_1 and t_2 are not separated by $\hat{L}$". Then $\hat{L}''_{K_1}$ and $\hat{L}''_{K_2}$ must intersect each other (this observation is the core of the proof). Together they define a path in $\hat{L}''$ connecting B_1 to B_2, so (due to Theorem 3.1) one of the two blobs must be special, hence t_1 or t_2 is in R_1, a contradiction.

Finally, consider the subcase where $K_1 \neq K_2$ and K_1 contains K_2 in one of its faces F. Then $\hat{L}_{K_2}$ is also contained in F (because it does not cross any skeleton cycle other than K_2). Then $\hat{L}''_{K_2}$ is also contained in F (by Theorem 6.1, Part 3). The only possibility for it not to separate t_1 from t_2 is if t_1 is in face F and $\hat{L}''_{K_2}$ also encloses t_1. Note that $\hat{L}''_{K_2}$ still intersects B_2. Then, we put t_1 in R_2.

It only remains to bound the weight of R_2. We use the fact that a candidate ear was not added in the ear step, to infer a bound on the weight of t_1. No part of $\hat{L}''_{K_2}$ or $\hat{L}_{K_2}$ was added in the ear step as a K_2-to-K_2 path separating t_1 from t_∞. Yet $\hat{L}''_{K_2}$ would have given a candidate ear (since by definition of K_1, we know that the clusters enclosed by K_2 do not separate t_1 from t_∞.) Since it was not selected, it must be that weight$(t_1) \leq \epsilon^3$length$(\hat{L}''_{K_2})$. We bound as follows the number of such terminals added to R_2 for each K_2. Notice that K_1 is the parent of K_2 in the nesting forest, t_1 is a terminal in the same face F of K_1 as K_2, and $C(t_1)$ encloses F. All terminals added to R_2 due to K_2 are part of $\tilde{T}_F$ by definition. Lemma 5.3 bounds their number to be at most ϵ^{-2}, hence the total weight of R_2 is at most $\epsilon\,$length$(\hat{L}'')$. ☐

Theorem 6.3. *The call* Spanner (G, v_∞, H) *returns a subgraph* $\widehat{G}$ *such that*

1. *length*$(\widehat{G}) \leq 2^{poly(1/\epsilon)}W$*, and*
2. $\widehat{G}$ *contains a multiway cut for* G *whose length is at most* $\mathsf{OPT}(G) + c\epsilon W$

Proof. Consider the set of edges formed by $\hat{L}^{(3)} = \hat{L}'' \cup \bigcup\{\mathsf{mincut}(t) : t \in \hat{R}'\}$. From Theorem 6.2, it follows that $\hat{L}^{(3)}$ is a feasible solution. Let us prove that it is near-optimal. By Theorem 6.1 and Inequality 3.4, the length of $\hat{L}''$ is at most $(1+O(\epsilon))$length$(\hat{L})+\frac{O(\epsilon^{-3.5})\kappa}{\theta}$length$(H)$ where H is the skeleton. By Theorem 5.4, $\hat{L}$ is near-optimal. Theorem 5.2 bounds length of H by $O(\epsilon^{-9}W)$. By the bound on the weight of $\hat{R}'$ in Theorem 6.2, $\sum_{t\in\hat{R}'} \mathsf{mincut}(t) = O(\epsilon)W$. Therefore, overall $\hat{L}^{(3)}$ has length $\mathsf{OPT}(G) + O(\epsilon)W$.

Finally, we bound the length of the spanner itself. Theorem 5.2 bounds the length of the skeleton H by $\epsilon^{-9}W$. By Inequality 3.4, the total length of mortar graphs is $O(\epsilon^{-1})$length(H). Fix a brick B. By Lemma 5.3, there are at most ϵ^{-2} terminals considered by the spanner algorithm when dealing with B. There are θ portals, so the number of configurations is bounded by $2^{\text{poly}(\theta)} = 2^{poly(1/\epsilon)}$. Each solution has length at most the length of the boundary of the brick. Therefore, the sum of lengths of all solutions for a brick B is $2^{poly(1/\epsilon)}$ times the length of the brick boundary. Altogether the length of the spanner is $2^{poly(1/\epsilon)}W$. ☐

Combining Theorem 6.3 with Lemma 4.4 yields Theorem 2.1.

References

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

[2] M. Bateni, M. Hajiaghayi, and D. Marx, *Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth*, in Proceedings of the Forty-Second Annual ACM Symposium on Theory of Computing (STOC'10), New York, NY, USA, 2010, ACM. to appear.

[3] C. Bentz, *A simple algorithm for multicuts in planar graphs with outer terminals*, Discrete Appl. Math., 157 (2009), pp. 1959–1964.

[4] G. Borradaile, C. Kenyon-Mathieu, and P. N. Klein, *A polynomial-time approximation scheme for Steiner tree in planar graphs*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1285–1294.

[5] G. Borradaile, P. N. Klein, and C. Mathieu, *Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon*, in Proceedings of the 10th Workshop on Algorithms and Data Structures, vol. 4619 of LNCS, 2007, pp. 275–286.

[6] ——, *A polynomial-time approximation scheme for Euclidean Steiner forest*, in Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science, 2008.

[7] ——, *A polynomial-time approximation scheme for steiner tree in planar graphs*, ACM Transactions on Algorithms, 5 (2009). Special Issue on SODA 2007.

[8] G. Calinescu, H. Karloff, and Y. Rabani, *An improved approximation algorithm for multiway cut*, in Proceedings of the 30th Symposium on the Theory of Computing, 1998, pp. 48–52.

[9] D. Z. Chen and X. Wu, *Efficient algorithms for k-terminal cuts on planar graphs*, Algorithmica, 38 (2004), pp. 299–316. Twelfth Annual International Symposium of Algorithms and Computation.

[10] W. Cunningham and L. Tang, *Optimal 3-terminal cuts and linear programming*, in Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization, no. 1610 in LNCS, 1999, pp. 114–125.

[11] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multiterminal cuts*, SIAM Journal on Computing, 23 (1994), pp. 864–894.

[12] D. Eppstein, *Subgraph isomorphism in planar graphs and related problems*, Journal of Graph Algorithms and Applications, 3 (1999), pp. 1–27.

[13] R. Erickson, C. Monma, and A. Veinott, *Send-and-split method for minimum-concave-cost network flows*, Mathematics of Operating Research, 12 (1987), pp. 634–664.

[14] D. Hartvigsen, *The planar multiterminal cut problem*, Discrete Appl. Math., 85 (1998), pp. 203–222.

[15] T. C. Hu, *Integer Programming and Network Flows*, Addison-Wesley, 1969.

[16] Y. Kamidoi, N. Yoshida, and H. Nagamochi, *A deterministic algorithm for finding all minimum k-way cuts*, SIAM J. Comput., 36 (2006/07), pp. 1329–1341 (electronic).

[17] D. R. Karger, P. N. Klein, C. Stein, M. Thorup, and N. E. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, in Proceedings of the thirty-first annual ACM Symposium on Theory of Computing, ACM, ed., New York, NY, USA, 1999, ACM Press, pp. 668–678.

[18] P. N. Klein, *A linear-time approximation scheme for planar weighted TSP*, in Proceedings of the Forty-Sixth Annual IEEE Symposium on Foundations of Computer Science, 2005, pp. 647–656.

[19] ———, *A subset spanner for planar graphs, with application to subset TSP*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006, pp. 749–756.

[20] ———, *A linear-time approximation scheme for tsp in undirected planar graphs with edge-weights*, SIAM J. Comput., 37 (2008), pp. 1926–1952.

[21] J. K. Park and C. A. Phillips, *Finding minimum-quotient cuts in planar graphs*, in STOC, 1993, pp. 766–775.