Approximation Schemes for Steiner Forest on Planar Graphs and Graphs of Bounded Treewidth*

MohammadHossein Bateni†
Department of Computer Science
Princeton University
Princeton, NJ, USA
mbateni@cs.princeton.edu

MohammadTaghi Hajiaghayi‡
AT&T Labs–Research
Florham Park, NJ, USA
hajiagha@research.att.com

Dániel Marx§
School of Computer Science
Tel Aviv University
Tel Aviv, Israel
dmarx@cs.bme.hu

ABSTRACT

We give the first *polynomial-time approximation scheme* (PTAS) for the *Steiner forest* problem on planar graphs and, more generally, on graphs of bounded genus. As a first step, we show how to build a *Steiner forest spanner* for such graphs. The crux of the process is a clustering procedure called *prize-collecting clustering* that breaks down the input instance into separate subinstances which are easier to handle; moreover, the terminals in different subinstances are far from each other. Each subinstance has a relatively inexpensive Steiner tree connecting all its terminals, and the subinstances can be solved (almost) separately. Another building block is a PTAS for *Steiner forest* on graphs of bounded treewidth. Surprisingly, *Steiner forest* is NP-hard even on graphs of treewidth 3. Therefore, our PTAS for bounded treewidth graphs needs a nontrivial combination of approximation arguments and dynamic programming on the tree decomposition. We further show that *Steiner forest* can be solved in polynomial time for series-parallel graphs (graphs of treewidth at most two) by a novel combination of dynamic programming and minimum cut computations, completing our thorough complexity study of *Steiner forest* in the range of bounded treewidth graphs, planar graphs, and bounded genus graphs.

Categories and Subject Descriptors

F.2.2 [**ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY**]: Nonnumerical Algorithms and Problems—*Computation on Discrete Structures*; G.1 [**NUMERICAL ANALYSIS**]; G.2 [**DISCRETE MATHEMATICS**]

General Terms

Algorithms, Design, Performance, Theory

1. INTRODUCTION

One of the most fundamental problems in combinatorial optimization and network design with both practical and theoretical significance is the Steiner forest problem, in which given a weighted graph $G = (V,E)$ and a set consisting of pairs of terminals, called *demands*, $\mathcal{D} = \{(s_1,t_1),(s_2,t_2),\ldots,(s_k,t_k)\}$, the goal is to find a minimum-cost forest F of G such that every pair of terminals in $\mathcal{D}$ is connected by a path in F. The first and the best approximation factor for this problem is 2 due to Agrawal, Klein and Ravi [1] (see also Goemans and Williamson [17]). The conference version of the Agrawal, Klein and Ravi [2] appeared in 1991, and there have been no improved approximation algorithms invented for Steiner forest. Recently Borradaile, Klein and Mathieu [11] obtain a Polynomial Time Approximation Scheme (PTAS) for Euclidean Steiner forest where the terminals are in the Euclidean plane. They pose obtaining a PTAS for Steiner forest in planar graphs, the natural generalization of Euclidean Steiner forest, as the main open problem. We note that in network design, planarity is a natural restriction, since in practical scenarios of physical networking, with cable or fiber embedded in the ground, crossings are rare or nonexistent. In this paper, we settle this open problem by obtaining a PTAS for planar graphs (and more generally, for bounded genus graphs) via a novel technique of *prize-collecting clustering* with potential use to obtain other PTASs in planar graphs.

The special case of the Steiner forest problem when all pairs have a common terminal is the classical Steiner tree problem, one of the first problems shown NP-hard by Karp [19]. The problem remains hard even on planar graphs [15]. In contrast to Steiner forest, a long sequence of papers give approximation factors better than 2 for this problem [29, 30, 7, 31, 25, 20, 18, 27]; the current best approximation ratio is 1.55 [27]. Since the problem is APX-hard in general graphs [8, 28], we do not expect to obtain a PTAS for this problem in general graphs. However, for the Euclidean Steiner tree

*For the omitted proofs and further discussion, refer to the full version of the paper [6].

†The author was supported by NSF ITR grants CCF-0205594, CCF-0426582 and NSF CCF 0832797, NSF CAREER award CCF-0237113, MSPA-MCS award 0528414, NSF expeditions award 0832797, as well as a Gordon Wu fellowship.

‡He is also with the Department of Computer Science, University of Maryland, College Park, MD, USA.

§He is supported by ERC Advanced Grant DMMCA.

STOC'10, June 5–8, 2010, Cambridge, Massachusetts, USA.

problem, the classic works of Arora [5] and Mitchell [24] present a PTAS. Obtaining a PTAS for Steiner tree on planar graphs, the natural generalization of Euclidean Steiner tree, remained a major open problem since the conference version of Arora [4] in 1996. Only in 2007, Borradaile, Mathieu, and Klein [12] settle this problem with a nice technique of constructing *light spanners* for Seiner trees in planar graphs. In this paper, we also generalize this result to obtain light spanners for Steiner forests.

Most approximation schemes for planar graph problems use (implicitly or explicitly) the fact that the problem is easy to solve on bounded-treewidth graphs (in fact, Demaine et al. [13] provide a general method of reducing many optimization problems on planar graphs to bounded-treewidth graphs). In particular, a keystone blackbox in the algorithm of Borradaile et al. [12] for Steiner tree is the result that, for every fixed value of k, the problem is polynomial-time solvable on graphs having treewidth at most k. There is a vast literature on algorithms for bounded-treewidth graphs and in most cases polynomial-time (or even linear-time) solvability follows from the well-understood standard technique of dynamic programming on tree decompositions. However, for Steiner *forest,* the obvious way of using dynamic programming does not give a polynomial-time algorithm. The difficulty is that, unlike in Steiner tree, a solution of Steiner forest induces a partition on the set of terminals and a dynamic programming algorithm needs to keep track of exponentially many such partitions. In fact, this approach seems to fail even for series-parallel graphs (that have treewidth at most 2); the complexity of the problem for series-parallel graphs was stated as an open question by Richey and Parker [26] in 1986. We resolve this question by giving a polynomial-time algorithm for Steiner forest on series-parallel graphs. The main idea is that even though algorithms based on dynamic programming have to evaluate subproblems corresponding to exponentially many partitions, the function describing these exponentially many values turn out to be submodular and it can be represented in a compact way by the cut function of a directed graph. On the other hand, Steiner forest becomes NP-hard on graphs of treewidth at most 3 [16]. Thus perhaps this is the first example when the complexity of a natural problem changes as treewidth increases from 2 to 3. In light of this hardness result, we investigate the approximability of the problem on bounded-treewidth graphs and show that, for every fixed k, Steiner forest admits a PTAS on graphs of treewidth at most k. The main idea of the PTAS is that if the dynamic programming algorithm considers only an appropriately constructed polynomial-size subset of the set of all partitions, then this produces a solution close to the optimum. Very roughly, the partitions in this subset are constructed by choosing a set of center points and classifying the terminals according to the distance to the center points. Our PTAS for planar graphs (and more generally, for bounded genus graphs) uses this PTAS for bounded-treewidth graphs. This completes our thorough study of Steiner forest in the range of bounded treewidth graphs, planar graphs and bounded genus graphs.

1.1 Our results and techniques

Our main result in this paper is a PTAS for the planar Steiner forest problem.

THEOREM 1. *For any constant $\bar{\varepsilon} > 0$, there is a polynomial-time $(1+\bar{\varepsilon})$-approximation algorithm for the Steiner forest problem on planar graphs and, more generally, on graphs of bounded genus.*

To this end, we build a *Steiner forest spanner* for the input graph and the set of demands; this is done in two steps. Roughly speaking, a Steiner forest spanner is a subgraph of the given graph whose cost is no more than a constant factor times the cost of the optimal Steiner forest, and furthermore, it contains a nearly optimal Steiner forest. Denote by $\mathrm{OPT}_{\mathcal{D}}(G)$ the minimum cost of a Steiner forest of G satisfying (connecting) all the demands in $\mathcal{D}$. We sometimes use OPT instead of $\mathrm{OPT}_{\mathcal{D}}(G)$. A subgraph H of G is a *Steiner forest spanner* with respect to demand set $\mathcal{D}$ if it has the following two properties:

Spanning Property: There is a forest in H that connects all demands in $\mathcal{D}$ and has length at most $(1+\varepsilon)OPT_{\mathcal{D}}(G)$, namely, $\mathrm{OPT}_{\mathcal{D}}(H) \leq (1+\varepsilon)\mathrm{OPT}_{\mathcal{D}}(G)$.

Shortness Property: The total length of H is not more than $f(\varepsilon) \cdot OPT_{\mathcal{D}}(G)$.

THEOREM 2. *Given any fixed $\varepsilon > 0$, a bounded genus graph $G_{in}(V_{in}, E_{in})$ and demand pairs $\mathcal{D}$, we can compute in polynomial time a Steiner forest spanner H for G_{in} with respect to demand set $\mathcal{D}$.*

The algorithm that we propose achieves this in time $O(n^2 \log n)$. The entire algorithm for Steiner forest runs in polynomial time but the exponent of the polynomial depends on ε and the genus of the input graph.

The proof of Theorem 2 heavily relies on a novel clustering method presented in Theorem 3 that allows us to (almost) separately build the spanners for smaller and far apart sets of demands. The rest of the spanner construction—done separately for each of the sets—uses ideas of Borradaile et al. [12], although there are still several technical differences. The clustering technique works for general graphs as opposed to the rest of the construction which requires the graphs to have bounded genus.

THEOREM 3. *Given an $\varepsilon > 0$, a graph $G_{in}(V_{in}, E_{in})$, and a set $\mathcal{D}$ of pairs of vertices, we can compute in polynomial time a set of trees $\{T_1, \ldots, T_k\}$, and a partition of demands $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$, with the following properties.*

1. *All the demands are covered, i.e., $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$.*
2. *All the terminals in $\mathcal{D}_i$ are spanned by the tree T_i.*
3. *The sum of the costs of all the trees T_i is no more than $(\frac{4}{\varepsilon} + 2)OPT_{\mathcal{D}}(G_{in})$.*
4. *The sum of the costs of minimum Steiner forests of all demand sets $\mathcal{D}_i$ is no more than $1+\varepsilon$ times the cost of a minimum Steiner forest of G_{in}; i.e., $\sum_i OPT_{\mathcal{D}_i}(G_{in}) \leq (1+\varepsilon)OPT_{\mathcal{D}}(G_{in})$.*

The last condition implies that (up to a small factor) it is possible to solve the demands $\mathcal{D}_i$ separately. Notice that this may lead to paying for portions of the solution more than once.

We will prove Theorem 3 in Section 3. Roughly speaking, the algorithm here first recognizes some connected components by running a 2-approximation Steiner forest algorithm. Clearly, this construction satisfies all but the last condition of the theorem. However, at this point, these connected components might not be sufficiently far from each other such that we can consider them separately. To fix this, we contract each connected component into a "super vertex" to which we assign a prize (potential) that is proportional to the sum of the edge weights of the corresponding component. Now we run an algorithm that we call *prize-collecting clustering*. The algorithm as well as some parts of its analysis bears similarities to a primal-dual method due to Agrawal, Klein and Ravi [2] and Goemans and Williamson [17]. Indeed our analysis strengthens these previous approaches by proving some local guarantees (instead of the global guarantee provided in these algorithms). In some sense, this clustering algorithm can also be seen as a generalization of an implicit clustering algorithm of Archer, Bateni, Hajiaghayi, and Karloff [3] who improve the best approximation factor for prize-collecting Steiner tree to $2-\varepsilon$, for some constant $\varepsilon > 0$. In this clustering, we consider a topological structure of the graph in which each edge is a curve connecting its endpoints whose length is equal to its weight. We color (portions of) edges by different colors each corresponding to a super vertex. These colors form a laminar family and the "depth" of each color is at most the prize

given to its corresponding super vertex. Using this coloring scheme we further connect some of these super vertices to each other with a cost proportional to the sum of their prizes. At the end, we show that now we can consider these combined connected components as separate clusters and, roughly speaking, an optimum solution need not connect two different clusters because of the concept of depth. We believe the prize-collecting clustering presented in this paper might have applications for other problems (esp. to obtain PTASs).

To obtain a PTAS as promised in Theorem 1, we first construct a spanner Steiner forest based on Theorem 2. On this spanner, we utilize a technique due to Klein [21], and Demaine, Hajiaghayi and Mohar [13] that reduces the problem of obtaining a PTAS in a planar (and more generally, bounded genus) graph whose total edge weight is within a constant factor of the optimum solution to that of finding an optimal solution in a graph of bounded treewidth (see the proof of Theorem 1 in Section 4 for more details.) However, there are no known polynomial-time algorithms in the literature for Steiner forest on bounded-treewidth graphs, so we cannot plug in such an algorithm to complete our PTAS. Therefore, we need to investigate Steiner forest on bounded-treewidth graphs.

Resolving an open question of Richey and Parker [26] from 1986, we design a polynomial-time algorithm for Steiner forest on series-parallel graphs. Very recently, using completely different techniques, a polynomial-time algorithm was presented for the special case of outerplanar graphs [16].

THEOREM 4. *The Steiner forest problem can be solved in polynomial time for subgraphs of series-parallel graphs (i.e., graphs of treewidth at most 2).*

Surprisingly, it turns out that the problem becomes NP-hard on graphs of treewidth at most 3 [16]. In the full version [6] we give a (different) NP-hardness proof, that highlights how submodularity (and hence the approach of Theorem 4) breaks if treewidth is 3. There exist a few known problems that are polynomial-time solvable for trees but NP-hard for graphs of treewidth 2 [32, 22, 23, 26], but to our knowledge, this is the first natural example where there is a complexity difference between treewidth 2 and 3.

Not being able to solve Steiner forest optimally on bounded-treewidth graphs is not an unavoidable obstacle for obtaining a PTAS on planar graphs: the technique of Klein [21], and Demaine, Hajiaghayi and Mohar [13] can still be applied when we have a PTAS for graphs of bounded treewidth. In Section 5, we demonstrate such a PTAS:

THEOREM 5. *For every fixed $w \geq 1$ and $\varepsilon > 0$, there is a polynomial-time $(1+\varepsilon)$-approximation algorithm for Steiner Forest on graphs with treewidth at most w.*

Note that the exponent of the polynomial in Theorem 5 depends on both ε and w; it remains an interesting question for future research whether this dependence can be removed.

The main idea of the PTAS of Theorem 5 is to reduce the set of partitions considered in the dynamic programming algorithm to a polynomially bounded subset, in a way that an $(1+\varepsilon)$-approximate solution using only these partitions is guaranteed to exist. The implementation of this idea consists of three components. First, we have to define which partitions belong to the polynomially bounded subset. These partitions are defined by choosing a bounded number of center points and a radius for each center. A terminal is classified into a class of the partition based on which center points cover it. Second, we need an algorithm that finds the best solution using only the allowed subset of partitions. This can be done following the standard dynamic programming paradigm, but the technical details are somewhat tedious. Third, we have to argue that there is a $(1+\varepsilon)$-approximate solution using only the allowed partitions. We show this by proving that if there is a solution that uses partitions that are not allowed, then it can be modified, incurring only a small increase in the cost, such that it uses only allowed partitions. The main argument here is that for each partition appearing in the solution, we try to select suitable center points. If these center points do not generate the required partition, then this means that a terminal is misclassified, which is only possible if the terminal is close to a center point. In this case, we observe that two components of the solution are close to each other and we can join them with only a small increase in the cost. The crucial point of the proof is a delicate charging argument, making use of the structure of bounded-treewidth graphs, which shows that repeated applications of this step results in a total increase that is not too large.

2. BASIC DEFINITIONS

Let $G(V,E)$ be a graph. As is customary, let $\delta(V')$ denote the set of edges having one endpoint in a subset $V' \subseteq V$ of vertices. For a subset of vertices $V' \subseteq V$, the subgraph of G induced by V' is denoted by $G[V']$. With slight abuse of notation, we sometimes use the edge set to refer to the graph itself. Hence, the above-mentioned subgraph may also be referred to by $E[V']$ for simplicity. We denote the length of a shortest x-to-y path in G as $\text{dist}_G(x,y)$. For an edge set E, we denote by $\ell(E) := \sum_{e \in E} c_e$ the total length of edges in E.

A collection $\mathcal{S}$ is said to be *laminar* if and only if for any two sets $C_1, C_2 \in \mathcal{S}$, we have $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$. Suppose $\mathcal{C}$ is a partition of a ground set V. Then, $\mathcal{C}(v)$ denotes for each $v \in V$ the set $C \in \mathcal{C}$ that contains v.

Given an edge $e = (u,v)$ in a graph G, the *contraction* of e in G denoted by G/e is the result of unifying vertices u and v in G, and removing all loops and multiple edges except the shortest edge. The contraction G/E' is defined as the result of iteratively contracting all the edges of E' in G, i.e., $G/E' := G/e_1/e_2/\ldots/e_k$ if $E' = \{e_1, e_2, \ldots, e_k\}$. Clearly, the planarity of G is preserved after the contraction. Similarly, contracting edges does not increase the cost of an optimal Steiner forest.

Finally we often use well-known concepts of *treewidth*, *nice tree-decomposition* and *2-cell embedding* in this paper. Due to lack of space, we refer the reader to the full version of the paper or standard graph theory texts such as [14] to see the exact definitions of these terms.

3. PRIZE-COLLECTING CLUSTERING

In this section, we describe an algorithm PC-CLUSTERING that is used to prove Theorem 3. The algorithm as well as its analysis bears similarities to the primal-dual method due to Agrawal, Klein and Ravi [1] and Goemans and Williamson [17]. It uses a technique that we call *prize-collecting clustering* and our analysis strengthens the previous approaches by proving some local guarantees (instead of the global guarantee provided in previous algorithms). We build a forest F_2 each of whose components correspond to one T_i sought in Theorem 3. Along the way, we also come up with a vector y satisfying the sets of constraints (1)-(3) below. During the process, we maintain a vector y satisfying all these constraints, and at the end, it will be true that all the constraints (2) hold with equality. The analysis takes advantage of the connection between F_2 and y.

We start with a 2-approximate[1] solution F^* satisfying all the demands in $\mathcal{D}$; the cost of F^* is at most 2OPT. The forest F^* consists of tree components T_i^*. In the following, we connect some of these components to make the trees T_i. It is easy to see that this construction guarantees the first two conditions of Theorem 3. We work on a graph $G(V,E)$ formed from G_{in} by contracting each tree component of F^*. A potential ϕ_v is associated with each vertex v of G, which is $\frac{1}{\varepsilon}$ times the cost of the tree component of F^* corresponding to v in case v is the contraction of a tree component, and zero

[1]Such a solution can be found via Goemans-Williamson's Steiner forest algorithm, for instance.

otherwise. We produce a solution to the following system of linear equations, where $y_{S,v}$ is defined for any $v \in S \subseteq V$, and c_e denotes the length of the edge e.

$$\sum_{S:e\in\delta(S)} \sum_{v\in S} y_{S,v} \le c_e \qquad \forall e \in E, \tag{1}$$

$$\sum_{S\ni v} y_{S,v} \le \phi_v \qquad \forall v \in V, \tag{2}$$

$$y_{S,v} \ge 0 \qquad \forall v \in S \subseteq V. \tag{3}$$

These constraints are very similar to the dual LP for the *prize-collecting Steiner tree* problem when ϕ_v are thought of as penalty values corresponding to the vertices. In the standard linear program for the prize-collecting Steiner tree problem, there is a special *root* vertex to which all the terminals are to be connected. Then, no set containing the root appears in the formulation.

The solution is built up in two stages. First we perform an *unrooted growth* to find a forest F_1 and a corresponding y vector. In the second stage, we *prune* some of the edges of F_1 to get another forest F_2. Uncontracting the trees T_i^* turns F_2 into the Steiner trees T_i in the statement of Theorem 3. Below we describe the two phases of Algorithm 1 (PC-CLUSTERING).

Growth.

We begin with a zero vector y, and an empty set F_1. We maintain a partition $\mathcal{C}$ of vertices V into clusters; it initially consists of singleton sets. Each cluster is either *active* or *inactive*; the cluster $C \in \mathcal{C}$ is *active* if and only if $\sum_{C'\subseteq C} \sum_{v\in C'} y_{C',v} < \sum_{v\in C} \phi_v$. A vertex v is *live* if and only if $\sum_{C\ni v} y_{C,v} < \phi_v$. Equivalently, a cluster $C \in \mathcal{C}$ is active if and only if there is a live vertex $v \in C$. We simultaneously *grow* all the active clusters by η. In particular, if there are $\kappa(C) > 0$ live vertices in an active cluster C, we increase $y_{C,v}$ by $\eta/\kappa(C)$ for each live vertex $v \in C$. Hence, y_C defined as $\sum_{v\in C} y_{C,v}$ is increased by η for an active cluster C. We pick the largest value for η that does not violate any of the constraints in (1) or (2). Obviously, η is finite in each iteration because the values of these variables cannot be larger than $\sum_v \phi_v$. Hence, at least one such constraint goes tight after the growth step. If this happens for an edge constraint for $e = (u,v)$, then there are two clusters $C_u \ni u$ and $C_v \ni v$ in $\mathcal{C}$; at least one of the two is growing. We merge the two clusters into $C = C_u \cup C_v$ by adding the edge e to F_1, remove the old clusters and add the new one to $\mathcal{C}$. Nothing needs to be done if a constraint (2) becomes tight. The number of iterations is at most $2|V|$, because at each event either a vertex dies, or the size of $\mathcal{C}$ decreases.

We can think of the growth stage as a process that colors portions of the edges of the graph. This gives a better intuition to the algorithm, and makes several of the following lemmas intuitively simple. 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. Suppose a cluster C is growing by an amount η. This is distributed among all the live vertices $v \in C$, where $y_{C,v}$ is increased by $\eta' := \eta/\kappa(C)$. As a result, we color by color v a connected portion with length η' of all the edges in $\delta(C)$. Finally, each edge e gets exactly $\sum_{C:e\in\delta(C)} y_{C,v}$ units of color v. We can perform a clean-up process, such that all the portions of color v are consecutive on an edge.[2] Hence, as a cluster expands, it colors its boundary by the amount of growth. At the time when two clusters merge, their colors barely touch each other. At each point in time, the colors associated with the vertices of a cluster form a connected region.

[2] We can do without the clean-up if we perform the coloring in a lazy manner. That is, we do not do the actual color assignment until the edge goes tight or the algorithm terminates. At this point, we go about putting colors on the edges, and we make sure the color corresponding to any pair (S,v) forms a consecutive portion of the edge. This property is not needed as part of our algorithm, though, and is merely for the sake of having a nice coloring which is of independent interest.

Pruning.

Let $\mathcal{S}$ contain every set that is a cluster at some point during the execution of the growth step. It can be easily observed that the clusters $\mathcal{S}$ are laminar and the maximal clusters are the clusters of $\mathcal{C}$. In addition, notice that $F_1[C]$ is connected for each $C \in \mathcal{S}$.

Let $\mathcal{B} \subseteq \mathcal{S}$ be the set of all such clusters that are tight, namely, for each $S \in \mathcal{B}$, we have $\sum_{S'\subseteq S} \sum_{v\in S'} y_{S',v} = \sum_{v\in S} \phi_v$. In the pruning stage, we iteratively remove some edges from F_1 to obtain F_2. More specifically, we first initialize F_2 with F_1. Then, as long as there is a cluster $S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$, we remove the edge e from F_2.

A cluster C is called a *pruned cluster* if it is pruned in the second stage in which case, $\delta(C) \cap F_2 = \emptyset$. Hence, a pruned cluster cannot have non-empty and proper intersection with a connected component of F_2.

Algorithm 1 PC-CLUSTERING

Input: planar graph $G_{in}(V_{in}, E_{in})$, and set of demands $\mathcal{D}$.
Output: set of trees T_i with associated $\mathcal{D}_i$.

1: Use the algorithm of Goemans and Williamson [17] to find a 2-approximate Steiner forest F^* of $\mathcal{D}$, consisting of tree components $T_1^*, \ldots, T_k^*$.
2: Contract each tree T_i^* to build a new graph $G(V,E)$.
3: For any $v \in V$, let ϕ_v be $\frac{1}{\xi}$ times the cost of the tree T_i^* corresponding to v, and zero if there is no such tree.
4: Let $F_1 \leftarrow \emptyset$.
5: Let $y_{S,v} \leftarrow 0$ for any $v \in S \subseteq V$.
6: Let $\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V\}$.
7: **while** there is a live vertex **do**
8: Let η be the largest possible value such that simultaneously increasing y_C by η for all active clusters C does not violate Constraints (1)-(3).
9: Let $y_{\mathcal{C}(v),v} \leftarrow y_{\mathcal{C}(v),v} + \frac{\eta}{\kappa(\mathcal{C}(v))}$ $\forall$ live vertices v.
10: **if** $\exists e \in E$ that is tight and connects two clusters **then**
11: Pick one such edge $e = (u,v)$.
12: Let $F_1 \leftarrow F_1 \cup \{e\}$.
13: Let $C \leftarrow \mathcal{C}(u) \cup \mathcal{C}(v)$.
14: Let $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{\mathcal{C}(u), \mathcal{C}(v)\}$.
15: Let $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$.
16: Let $F_2 \leftarrow F_1$.
17: Let $\mathcal{B}$ be the set of all clusters $S \in \mathcal{S}$ such that $\sum_{v\in S} y_{S,v} = \sum_{v\in S} \phi_v$.
18: **while** $\exists S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$ for an edge e **do**
19: Let $F_2 \leftarrow F_2 \setminus \{e\}$.
20: Construct F from F_2 by uncontracting all trees T_i^*.
21: Let F consist of tree components T_i.
22: Output the set of trees $\{T_i\}$, along with $\mathcal{D}_i := \{(s,t) \in \mathcal{D} : s,t \in V(T_i)\}$.

We first bound the cost of the forest F_2. The following lemma is similar to the analysis of the algorithm in [17]. However, we do not have a primal LP to give a bound on the dual. Rather, the upper bound for the cost is the sum of all the potential values $\sum_v \phi_v$. In addition, we bound the cost of a forest F_2 that may have more than one connected component, whereas the prize-collecting Steiner tree algorithm of [17] finds a connected graph at the end.

LEMMA 6. *The cost of F_2 is at most $2\sum_{v\in V} \phi_v$.*

PROOF. The strategy is to prove that the cost of this forest is at most $2\sum_{v\in S\subseteq V} y_{S,v} \le 2\sum_{v\in V} \phi_v$. The equality follows from Equation (2). Recall that the growth phase has several events corresponding to an edge or set constraint going tight. We first break

apart y variables by epoch. Let t_j be the time at which the j^{th} event point occurs in the growth phase ($0 = t_0 \le t_1 \le t_2 \le \cdots$), so the j^{th} epoch is the interval of time from t_{j-1} to t_j. For each cluster C, let $y_C^{(j)}$ be the amount by which $y_C := \sum_{v \in C} y_{C,v}$ grew during epoch j, which is $t_j - t_{j-1}$ if it was active during this epoch, and zero otherwise. Thus, $y_C = \sum_j y_C^{(j)}$. Because each edge e of F_2 was added at some point by the growth stage when its edge packing constraint (1) became tight, we can exactly apportion the cost c_e amongst the collection of clusters $\{C : e \in \delta(C)\}$ whose variables "pay for" the edge, and can divide this up further by epoch. In other words, $c_e = \sum_j \sum_{C : e \in \delta(C)} y_C^{(j)}$. We will now prove that the total edge cost from F_2 that is apportioned to epoch j is at most $2\sum_C y_C^{(j)}$. In other words, during each epoch, the total rate at which edges of F_2 are paid for by all active clusters is at most twice the number of active clusters. Summing over the epochs yields the desired conclusion.

We now analyze an arbitrary epoch j. Let $\mathcal{C}_j$ denote the set of clusters that existed during epoch j. Consider the graph F_2, and then collapse each cluster $C \in \mathcal{C}_j$ into a supernode. Call the resulting graph H. Although the nodes of H are identified with clusters in $\mathcal{C}_j$, we will continue to refer to them as clusters, in order to to avoid confusion with the nodes of the original graph. Some of the clusters are active and some may be inactive. Let us denote the active and inactive clusters in $\mathcal{C}_j$ by $\mathcal{C}_{act}$ and $\mathcal{C}_{dead}$, respectively. The edges of F_2 that are being partially paid for during epoch j are exactly those edges of H that are incident to an active cluster, and the total amount of these edges that is paid off during epoch j is $(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_{act}} \deg_H(C)$. Since every active cluster grows by exactly $t_j - t_{j-1}$ in epoch j, we have $\sum_C y_C^{(j)} \ge \sum_{C \in \mathcal{C}_j} y_C^{(j)} = (t_j - t_{j-1})|\mathcal{C}_{act}|$. Thus, it suffices to show that $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \le 2|\mathcal{C}_{act}|$.

First we must make some simple observations about H. Since F_2 is a subset of the edges in F_1, and each cluster represents a disjoint induced connected subtree of F_1, the contraction to H introduces no cycles. Thus, H is a forest. All the leaves of H must be alive, because otherwise the corresponding cluster C would be in $\mathcal{B}$ and hence would have been pruned away.

With this information about H, it is easy to bound $\sum_{C \in \mathcal{C}_{act}} \deg_H(C)$. The total degree in H is at most $2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|)$. Noticing that the degree of dead clusters is at least two, we get $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \le 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|) - 2|\mathcal{C}_{dead}| = 2|\mathcal{C}_{act}|$ as desired. □

The following lemma gives a sufficient condition for two vertices that end up in the same component of F_2. This is a corollary of our pruning rule which has a major difference from other pruning rules. Unlike the previous work, we do not prune the entire subgraph; rather, we only remove some edges, increasing the number of connected components.

LEMMA 7. *Two vertices u and v of V are connected via F_2 if there exist sets S, S' both containing u, v such that $y_{S,v} > 0$ and $y_{S',u} > 0$.*

PROOF. The growth stage connects u and v since $y_{S,v} > 0$ and $u, v \in S$. Consider the path p connecting u and v in F_1. All the vertices of p are in S and S'. For the sake of reaching a contradiction, suppose some edges of p are pruned. Let e be the first edge being pruned on the path p. Thus, there must be a cluster $C \in \mathcal{B}$ cutting e; furthermore, $\delta(C) \cap p = \{e\}$, since e is the first edge pruned from p. The laminarity of the clusters $\mathcal{S}$ gives $C \subset S, S'$, since C contains exactly one endpoint of e. If C contains both or no endpoints of p, it cannot cut p at only one edge. Thus, C contains exactly one endpoint of p, say v. We then have $\sum_{C' \subseteq C} y_{C',v} = \phi_v$, because C is tight. However, as C is a *proper* subset of S, this contradicts with $y_{S,v} > 0$, proving the supposition is false. The case C contains u is symmetric. □

Consider a pair (v, S) with $y_{S,v} > 0$. If subgraph G' of G has an edge that goes through the cut $(S, \bar{S})$, at least a portion of length $y_{S,v}$ of G' is colored with the color v due to the set S. Thus, if G' cuts all the sets S for which $y_{S,v} > 0$, we can charge part of the length of G' to the potential of v. Later in Lemma 10, we are going to use potentials as a lower bound on the optimal solution. More formally, we say a graph $G'(V, E')$ *exhausts a color u* if and only if $E' \cap \delta(S) \neq \emptyset$ for any $S : y_{S,u} > 0$. The proof of the following corollary is omitted here, however, it is implicit in the proof of Lemma 10 below. We do not use this corollary explicitly. Nevertheless, it gives insight into the analysis below.

COROLLARY 8. *If a subgraph H of G connects two vertices u_1, u_2 from different components of F_2 (which are contracted versions of the components in the initial 2-approximate solution), then H exhausts the color corresponding to at least one of u_1 and u_2.*

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

LEMMA 9. *Let L be the set of colors exhausted by subgraph G' of G. The cost of $G'(V, E')$ is at least $\sum_{v \in L} \phi_v$.*

This is quite intuitive. Recall that the y variables color the edges of the graph. Consider a segment on edges corresponding to cluster S with color v. At least one edge of G' *passes through* the cut $(S, \bar{S})$. Thus, a portion of the cost of G' can be charged to $y_{S,v}$. Hence, the total cost of the graph G' is at least as large as the total amount of colors paid for by L. Here is the formal proof.

PROOF. The cost of $G'(V, E)$ is

$$\begin{aligned}
\sum_{e \in E'} c_e &\ge \sum_{e \in E'} \sum_{S : e \in \delta(S)} y_S && \text{by (1)} \\
&= \sum_S |E' \cap \delta(S)| y_S \\
&\ge \sum_{S : E' \cap \delta(S) \neq \emptyset} y_S \\
&= \sum_{S : E' \cap \delta(S) \neq \emptyset} \sum_{v \in S} y_{S,v} \\
&= \sum_v \sum_{S \ni v : E' \cap \delta(S) \neq \emptyset} y_{S,v} \\
&\ge \sum_{v \in L} \sum_{S \ni v : E' \cap \delta(S) \neq \emptyset} y_{S,v} \\
&= \sum_{v \in L} \sum_{S \ni v} y_{S,v},
\end{aligned}$$

because $y_{S,v} = 0$ if $v \in L$ and $E' \cap \delta(S) = \emptyset$,

$$= \sum_{v \in L} \phi_v \qquad \text{by a tight version of (2).} \quad \square$$

Recall that the trees T_i^* of the 2-approximate solution F^* are contracted in F_2. Construct F from F_2 by uncontracting all these trees. Let F consist of tree components T_i. It is not difficult to verify that F is indeed a forest, but we do not need this condition since we can always remove cycles to find a forest. Define $\mathcal{D}_i := \{(s,t) \in \mathcal{D} : s, t \in V(T_i)\}$, and let OPT_i denote (the cost of) the optimal Steiner forest satisfying the demands $\mathcal{D}_i$.

LEMMA 10. $\sum_i OPT_i \le (1+\varepsilon) OPT$.

PROOF. If each tree component of OPT only satisfies demands from a single $\mathcal{D}_i$, we are done. Otherwise, we build a solution OPT' consisting of forests OPT'_i for each $\mathcal{D}_i$, such that $\sum_i \text{OPT}'_i \le (1+\varepsilon)\text{OPT}$. Then, $\text{OPT}_i \le \text{OPT}'_i$ finishes the argument. Instead of explicitly identifying the forests OPT'_i, we construct an *inexpensive* set of trees $\mathcal{T}$ where each $T \in \mathcal{T}$ is associated with a subset $D_T \subseteq \mathcal{D}$ of demands, such that

- $T \in \mathcal{T}$ connects all the demands in D_T,
- all the demands are satisfied, namely, $\cup_{T\in\mathcal{T}} D_T = \mathcal{D}$, and
- for each $T \in \mathcal{T}$, there exists a $\mathcal{D}_i$ where $D_T \subseteq \mathcal{D}_i$.

Then, each OPT'_i above is merely a collection of several trees in $\mathcal{T}$. We now describe how $\mathcal{T}$ is constructed. Start with the optimal solution OPT. Contract the trees T_i^* of the 2-approximate solution F^* to build a graph $\hat{\text{OPT}}$. Initially, the set $D_{\hat{T}}$ associated with each connected component $\hat{T}$ of $\hat{\text{OPT}}$ specifies the set of demands connected via $\hat{T}$; notice that the endpoints of any demand are both part of one super-node because F^* is feasible. If the color u corresponding to a component T_i^* is exhausted by $\hat{\text{OPT}}$, add T_i^* to $\mathcal{T}$, and let $D_{T_i^*}$ be the set of demands satisfied via T_i^*. Then, for every component $\hat{T}_i$ of $\hat{\text{OPT}}$, remove $\cup_{T\in\mathcal{T}} D_T$ from $D_{\hat{T}_i}$. Finally, construct the uncontracted version of all the components $\hat{T}_i$, and add them to $\mathcal{T}$.

The first condition above clearly holds by the construction. For the second condition, note that at the beginning all the demands are satisfied by the trees $\hat{T}_i$, and we only remove demands from $D_{\hat{T}_i}$ when they are satisfied elsewhere. The last condition is proved by contradiction. Suppose a tree $T \in \mathcal{T}$ satisfies demands from two different groups $\mathcal{D}_1$ and $\mathcal{D}_2$. Clearly, T cannot be one of the contracted trees T_i^*, since those trees only serve a subset of one $\mathcal{D}_i$. Hence, T must be a tree component of OPT. Let $u, v \in V$ be two vertices from different components of F_2 that are connected in T. Since u and v are not connected in F_2, Lemma 7 ensures that for at least one of these vertices, say u, we have $y_{S,u} = 0\ \forall S \ni u, v$. Thus, $y_{S,u} > 0$ implies $v \notin S$. As T connects u and v, this means T exhausts the color u. We reach a contradiction, because the demands corresponding to u are still in D_T.

Finally, we show that the cost of $\mathcal{T}$ is small. Recall that $\mathcal{T}$ consists of two kinds of trees: tree components of OPT as well as trees corresponding to exhausted colors of OPT. We claim the cost of the latter is at most εOPT. Let L be the set of exhausted colors of OPT. The cost of the corresponding trees is $\varepsilon\sum_{v\in L}\phi_v$ by definition of ϕ_v. Lemma 9, however, gives $\text{OPT} \geq \sum_{v\in L}\phi_v$. Therefore, the total cost of $\mathcal{T}$ is at most $(1+\varepsilon)$OPT. □

Now, we are ready to prove the main theorem of this section.

PROOF OF THEOREM 3. The first condition of the lemma follows directly from our construction: we start with a solution, and never disconnect one of the tree components in the process. The construction immediately implies the second condition. By Lemma 6, the cost of F_2 is at most $2\sum_{v\in V}\phi_v \leq \frac{4}{\varepsilon}\text{OPT}$. Thus, F costs no more than $(4/\varepsilon + 2)$OPT, giving the third condition. Finally, Lemma 10 establishes the last condition. □

4. PTAS FOR PLANAR STEINER FOREST

The full version of the paper [6] shows how to build a spanner for the input graph $G_{in}(V_{in}, E_{in})$ with respect to the demand set $\mathcal{D}$. From Theorem 3, we obtain a set of trees $\{T_1, \ldots, T_k\}$, associated with a partition of demands $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$: tree T_i connects all the demands $\mathcal{D}_i$, and the total cost of trees T_i is in $O(\text{OPT})$. The construction goes along the same lines as that of Borradaile et al. [12], yet there are certain differences, especially in the analysis. We separately build a graph H_i for each T_i, and finally let H be the union of all graphs H_i.

Having proved the spanner result, we can present our main PTAS for *Steiner forest* on planar graphs in this section. We first mention two main ingredients of the algorithm. We invoke the following result due to Demaine, Hajiaghayi and Mohar [13].

THEOREM 11 ([13]). *For a fixed genus g, and any integer $k \geq 2$, and for every graph G of Euler genus at most g, the edges of G can be partitioned into k sets such that contracting any one of the sets results in a graph of treewidth at most $O(g^2k)$. Furthermore, such a partition can be found in $O(g^{5/2}n^{3/2}\log n)$ time.*

As a corollary, this holds for a planar graph (which has genus zero).

We can now prove the main theorem of this paper. Algorithm 2 (PSF-PTAS) shows the steps of the PTAS.

Algorithm 2 PSF-PTAS

Input: planar graph $G_{in}(V_{in}, E_{in})$, and set of demands $\mathcal{D}$.
Output: Steiner forest F satisfying $\mathcal{D}$.

1: Construct the Steiner forest spanner H.
2: Let $k \leftarrow 2f(\varepsilon)/\bar{\varepsilon}$.
3: Let $\varepsilon \leftarrow \min(1, \bar{\varepsilon}/6)$.
4: Using Theorem 11, partition the edges of H into $E_1, \ldots, E_k$.
5: Let $i^* \leftarrow \arg\min_i \ell(E_i)$.
6: Find a $(1+\varepsilon)$-approximate Steiner forest F^* of $\mathcal{D}$ in H/E_{i^*} via Theorem 5.
7: Output $F^* \cup E_{i^*}$.

PROOF OF THEOREM 1. Given are planar graph G_{in}, and the set of demand pairs $\mathcal{D}$. We build a Steiner forest spanner H using Theorem 2. For a suitable value of k whose precise value will be fixed below, we apply Theorem 11 to partition the edges of H into $E_1, E_2, \ldots, E_k$. Let E_{i^*} be the set having the least total length. The total length of edges in E_{i^*} is at most $\ell(H)/k$. Contracting E_{i^*} produces a graph H^* of treewidth $O(k)$.

Theorem 5 allows us to find a solution OPT^* corresponding to H^*. Adding the edges E_{i^*} clearly gives a solution for H whose value is at most $(1+\varepsilon)\text{OPT}_{\mathcal{D}}(H) + \ell(H)/k$. Letting $\varepsilon = \min(1, \bar{\varepsilon}/6)$ and $k = 2f(\varepsilon)/\bar{\varepsilon}$ guarantees that the cost of this solution is

$$\begin{aligned} &\leq (1+\varepsilon)^2\text{OPT}_{\mathcal{D}}(G_{in}) + \ell(H)/k \\ &\leq (1+\varepsilon)^2\text{OPT}_{\mathcal{D}}(G_{in}) + \frac{\bar{\varepsilon}}{2}\text{OPT}_{\mathcal{D}}(G_{in}) \\ &(1+\bar{\varepsilon})\text{OPT}_{\mathcal{D}}(G_{in}). \quad \square \end{aligned}$$

The running time of all the algorithm except for the bounded-treewidth PTAS is bounded by $O(n^2 \log n)$. The parameter k above has a singly exponential dependence on ε. Yet, the running time of the current procedure for solving the bounded-treewidth instances is not bounded by a low-degree polynomial; rather, k and ε appear in the exponent in the polynomial. Were we able to improve the running time of this procedure, we would obtain a PTAS that runs in time $O(n^2 \log n)$.

4.1 Extension to bounded-genus graphs

We can generalize this result to the case of bounded-genus graphs. Theorem 3 does not assume any special structure for the input graph. To construct a spanner in this case, we use the generalized ideas of Borradaile et al. [10]. This process does not increase the Euler genus of the graph, since the resulting graph has a subset of original edges. Theorem 11 works for such graphs as well, and hence, as in Theorem 1, we can reduce the problem to a bounded-treewidth graph on which we apply Theorem 5.

THEOREM 12. *For any constant $\varepsilon > 0$, there is a polynomial-time $(1+\varepsilon)$-approximation algorithm for the graphs of Euler genus at most g.*

5. PTAS FOR GRAPHS OF BOUNDED TREEWIDTH

The purpose of this section is to sketch the proof of Theorem 5. For definitions concerning tree decompositions, we refer the reader to the full version [6]. We define a notion of group that will be crucial in the algorithm. A group is defined by a set S of center vertices, a set X of "interesting" vertices, and a maximum distance r; the *group* $\mathcal{G}_G(X, S, r)$ contains S and those vertices of X that are at distance at most r from some vertex in S.

LEMMA 13. *Let T be a Steiner tree of $X \subseteq V(G)$ with cost W. For every $\varepsilon > 0$, there is a set $S \subseteq X$ of $O(1+1/\varepsilon)$ vertices such that $X = \mathcal{G}_G(X, S, \varepsilon W)$.*

Let $\mathcal{B} = (B_i)_{i=1\ldots n}$ be the bags of a rooted nice tree decomposition of width k. Let V_i be the set of vertices appearing in B_i or in a descendant of B_i. Let A_i be the set of *active vertices* at bag B_i: those vertices $v \in V_i$ for which there is a demand $\{v,w\} \in \mathcal{D}$ with $w \notin V_i$. Let $G_i := G[V_i]$. A Steiner forest F induces a partition $\pi_i(F)$ of A_i for every $i = 1,\ldots,n$: let two vertices of A_i be in the same class of $\pi_i(F)$ if and only if they are in the same component of F. Note that if F is restricted to G_i, then a component of F can be split into up to $k+1$ components, thus $\pi_i(F)$ is a coarser partition then the partition defined by the components of the restriction of F to G_i.

Let $\Pi = (\Pi_i)_{i=1\ldots n}$ be a collection such that Π_i is a set of partitions of A_i. If $\pi_i(F) \in \Pi_i$ for every bag B_i, then we say that F *conforms* to Π. The following lemma gives an algorithm for finding a solution whose cost is minimum among the solutions conforming to a given Π. The proof follows the standard dynamic programming approach, but is quite tedious and technical, see the full version [6].

LEMMA 14. *For every fixed k, there is a polynomial time algorithm that, given a graph G with treewidth at most k and a collection Π, finds the minimum cost Steiner forest conforming to Π.*

Next we construct a polynomial-size collection Π such that there is a $(1+\varepsilon)$-approximate solution that conforms to Π. Putting together these two results, we get a PTAS for the Steiner forest problem on bounded treewidth graphs. Recall that the collection Π contains a set of partitions Π_i for each $i \in I$. Each partition in Π_i is defined by a sequence $((S_1,r_1),\ldots,(S_p,r_p))$ of at most $k+1$ pairs and a partition ρ of $\{1,\ldots,p\}$. The pair (S_j,r_j) consists of a set S_j of $O((k+1)(1+1/\varepsilon))$ vertices of G_i and a nonnegative integer r_j, which equals the distance between two vertices of G. There are at most $|V(G)|^{O((k+1)(1+1/\varepsilon))} \cdot |V(G)|^2$ possible pairs (S_j,r_j) and hence at most $|V(G)|^{O((k+1)^2(1+1/\varepsilon))}$ different sequences. The number of possible partitions ρ is $O(k^k)$. Thus if we construct Π_i by considering all possible sequences constructed from every possible choice of (S_j,r_j), the size of Π_i is polynomial in $|V(G)|$ for every fixed k and ε.

We construct the partition π corresponding to a particular sequence and ρ the following way. Each pair (S_j,r_j) can be used to define a group $R_j = \mathcal{G}_G(A_i,S_j,r_j)$ of A_i. Roughly speaking, for each class P of ρ, there is a corresponding class of π that contains the union of R_j for every $j \in P$. However, the actual definition is somewhat more complicated. We want π to be a partition, which means that the subsets of A_i corresponding to the different classes of ρ should be disjoint. In order to ensure disjointness, we define $R'_j := R_j \setminus \bigcup_{j'=1}^{j-1} R_{j'}$. The partition π of A_i is constructed as follows: for each class P of ρ, we let $\bigcup_{j\in P} R'_j$ be a class of π. Note that these classes are disjoint by construction. If these classes fully cover A_i, then we put the resulting partition π into Π_i; otherwise, the sequence does not define a partition. This finishes the construction of Π_i.

Before showing that there is a good approximate solution conforming to the collection Π defined above, we need a further definition. For two vertices u and v, we denote by $u < v$ the fact that the topmost bag containing u is a proper descendant of the topmost bag containing v. Note that each bag is the topmost bag of at most one vertex in a nice tree decomposition (recall that we can assume that the root bag contains only a single vertex). Thus if u and v appear in the same bag, then $u < v$ or $v < u$ holds, i.e., this relation defines an ordering of the vertices in a bag. We can extend this relation to connected subset of vertices: for two disjoint connected sets K_1, K_2, $K_1 < K_2$ means that K_2 has a vertex v such that $u < v$ for every vertex $u \in K_1$, in other words, $K_1 < K_2$ means that the topmost bag where vertices from K_1 appear is a proper descendant of the topmost bag where vertices from K_2 appear. If there is a bag containing vertices from both K_1 and K_2, then either $K_1 < K_2$ or $K_2 < K_1$ holds. The reason for this is that the bags containing vertices from $K_1 \cup K_2$ form a connected subtree of the tree decomposition, and if the topmost bag in this subtree contains vertex $v \in K_1 \cup K_2$, then $u < v$ for every other vertex u in $K_1 \cup K_2$.

LEMMA 15. *There is a $(1+k\varepsilon)$-approximate solution conforming to Π.*

PROOF. Let F be a minimum cost Steiner forest. We describe a procedure that adds further edges to F to transform it into a Steiner forest F' that conforms to Π and has cost at most $(1+k\varepsilon)\ell(F)$. We need a delicate charging argument to show that the total increase of the cost is at most $k\varepsilon \cdot \ell(F)$ during the procedure. In each step, we charge the increase of the cost to an ordered pair (K_1,K_2) of components of F. We are charging only to pairs (K_1,K_2) having the property that $K_1 < K_2$ and there is a bag containing vertices from both K_1 and K_2. Observe that if B_i is the topmost bag where vertices from K_1 appear, then these properties imply that a vertex of K_2 appears in this bag as well. Otherwise, if every bag containing vertices of K_2 appears above B_i, then there is no bag containing vertices from both K_1 and K_2; if every bag containing vertices from K_2 appears below B_i, then $K_1 < K_2$ is not possible. Thus a component K_1 can be the first component of at most k such pairs (K_1,K_2): since the components are disjoint, the topmost bag containing vertices from K_1 can intersect at most k other components. We will charge a cost increase of at most $\varepsilon \cdot \ell(K_1)$ on the pair (K_1,K_2), thus the total increase is at most $k\varepsilon \cdot \ell(F)$. It is a crucial point of the proof that we charge on (pairs of) components of the original solution F, even after several modification steps, when the components of F' can be larger than the original components of F. Actually, in the proof to follow, we will refer to three different types of components:

(a) Components of the current solution F'.
(b) Each component of F' contains one or more components of F.
(c) If a component of F is restricted to the subset V_i, then it can split into up to $k+1$ components.

To emphasize the different meanings, and be clear as well, we use the terms a-component, b-component, and c-component.

Initially, we set $F' := F$ and it will be always true that F' is a supergraph of F, thus F' defines a partition of the b-components of F. Suppose that there is a bag B_i such that the partition $\pi_i(F')$ of A_i induced by F' is not in Π_i. Let $K_1 < K_2 < \cdots < K_p$ be the b-components of F intersecting B_i, ordered by the relation $<$. Some of these b-components might be in the same a-component of F'; let ρ be the partition of $\{1,\ldots,p\}$ defined by F' on these b-components.

Let $A_{i,j}$ be the subset of A_i contained in K_j. The intersection of b-component K_j with V_i gives rise to at most $k+1$ c-components, each of cost at most $\ell(K_j)$. Thus by Lemma 13, there is a set $S_j \subseteq V(K_j)$ of at most $O((k+1)(1+1/\varepsilon))$ vertices such that $A_{i,j} = \mathcal{G}_{G_i}(A_{i,j},S_j,r_j)$ for some $r_j \le \varepsilon \cdot \ell(K_j)$. If the sequence (S_1,r_1), $\ldots$, (S_p,r_p) and the partition ρ give rise to the partition $\pi_i(F')$, then $\pi_i(F') \in \Pi_i$. Otherwise, let us investigate the reason why this sequence and ρ do not define the partition $\pi_i(F')$. Let R_j, R'_j be defined as in the definition of Π_i, i.e., $R_j = \mathcal{G}_G(A_i,S_j,r_j)$ and $R'_j := R_j \setminus \bigcup_{j'=1}^{j-1} R_{j'}$. It is clear that $A_{i,j} \subseteq R_j$. Therefore, every vertex of A_i is contained in some R_j and hence in some R'_j. Thus the sequence does define a partition π, but maybe a partition different from $\pi_i(F')$. Let $\rho(j)$ be the class of ρ containing j. If for every $1 \le j \le p$, every vertex of $A_{i,j}$ is contained in $\bigcup_{j'\in\rho(j)} R'_{j'}$, then π and $\pi_i(F')$ are the same. So suppose

that some vertex $v \in A_{i,j}$ is not in $\bigcup_{j' \in \rho(j)} R'_{j'}$. As $v \in R_j$, this means that $v \in R_{j^*}$ for some $j^* < j$ and $j^* \notin \rho(j)$. The fact that $R_{j^*} = \mathcal{G}_{G_i}(A_i, S_{j^*}, r_{j^*})$ contains $v \in A_{i,j}$ means that there is a vertex $u \in S_{j^*}$ such that $d_{G_i}(u,v) \leq r_{j^*} \leq \varepsilon \cdot \ell(K_{j^*})$. Note that u is a vertex of b-component K_{j^*} (as $u \in S_{j^*}$ and by definition S_{j^*} is a subset of $V(K_{j^*})$) and v is a vertex of b-component K_j. We modify F' by adding a shortest path that connects u and v. Clearly, this increases the cost of F' by at most $\varepsilon \cdot \ell(K_{j^*})$, which we charge on the pair (K_{j^*}, K_j). Note that K_j and K_{j^*} both intersect the bag B_i and $K_{j^*} < K_j$, as required in the beginning of the proof. Furthermore, K_j and K_{j^*} are in the same a-component of F' after the modification, but not before. Thus we charge at most once on the pair (K_{j^*}, K_j).

Since the modification always extends F', the procedure described above terminates after a finite number of steps. At this point, every partition $\pi_i(F')$ belongs to the corresponding set Π_i, that is, the solution F' conforms to Π. □

6. ALGORITHM FOR SERIES-PARALLEL GRAPHS

A series-parallel graph can be built form elementary blocks using two operations: parallel connection and series connection. The algorithm of Theorem 4 uses dynamic programming on the construction of the series-parallel graph. For each subgraph arising in the construction, we find a minimum weight forest that connects some of the terminal pairs, connects a subset of the terminals to the "left exit point" of the subgraph, and connects the remaining terminals to the "right exit point" of the subgraph. The minimum weight depends on the subset of terminals connected to the left exit point, thus it seems that we need to determine exponentially many values (one for each subset). Fortunately, it turns out that the minimum weight is a submodular function of the subset. Furthermore, we show that this function can be represented by the cut function of a directed graph and this directed graph can be easily constructed if the directed graphs corresponding to the building blocks of the series-parallel subgraph are available. Thus, following the construction of the series-parallel graph, we can build all these directed graphs and determine the value of the optimum solution by the computation of a minimum cut.

Formally, a *series-parallel graph* $G(x,y)$ with distinguished vertices x,y is an undirected graph that can be constructed using the following rules:

- An edge xy is a series-parallel graph.
- If $G_1(x_1,y_1)$ and $G_2(x_2,y_2)$ are series-parallel graphs, then the graph $G(x,y)$ obtained by identifying x_1 with x_2 and y_1 with y_2 is a series-parallel graph with distinguished vertices $x := x_1 = x_2$ and $y := y_1 = y_2$ *(parallel connection).*
- If $G_1(x_1,y_1)$ and $G_2(x_2,y_2)$ are series-parallel graphs, then the graph $G(x,y)$ obtained by identifying y_1 with x_2 is a series-parallel graph with distinguished vertices $x := x_1$ and $y := y_2$ *(series connection).*

We prove Theorem 4 in this section by constructing a polynomial-time algorithm to solve *Steiner forest* on series-parallel graphs. It is well-known that the treewidth of a graph is at most 2 if and only if it is a subgraph of a series parallel graph [9]. Since setting the length of an edge to ∞ is essentially the same as deleting the edge, it follows that *Steiner forest* can be solved in polynomial time on graphs with treewidth at most 2.

Let $(G,\mathcal{D})$ be an instance of *Steiner forest* where G is series parallel. For $i = 1,\ldots,m$, denote by $G_i(x_i,y_i)$ all the intermediary graphs appearing in the series-parallel construction of G. We assume that these graphs are ordered such that $G = G_m$ and if G_i is obtained from G_{j_1} and G_{j_2}, then $j_1, j_2 < i$. Let $\mathcal{D}_i \subseteq \mathcal{D}$ contain those pairs $\{u,v\}$ where both vertices are in $V(G_i)$. Let A_i be those vertices $v \in V(G_i)$ for which there exists a pair $\{v,u\} \in \mathcal{D}$ with $u \notin V(G_i)$ (note that $A_m = \emptyset$ and $\mathcal{D}_m = \mathcal{D}$). For every G_i, we define two integer values a_i, b_i and a function f_i:

- Let a_i be the minimum cost of a solution F of the instance $(G_i, \mathcal{D}_i)$ with the additional requirements that x_i and y_i are connected in F and every vertex in A_i is in the same component as x_i and y_i.
- Let G'_i be the graph obtained from G_i by identifying vertices x_i and y_i. Let b_i be the minimum cost of a solution F of the instance $(G'_i, \mathcal{D}_i)$ with the additional requirement that A_i is in the same component as $x_i = y_i$.
- For every $S \subseteq A_i$, let $f_i(S)$ be the minimum cost of a solution F of the instance $(G_i, \mathcal{D}_i)$ with the additional requirements that x_i and y_i are not connected, every $v \in S$ is in the same component as x_i, and every $v \in A_i \setminus S$ is in the same component as y_i. (If there is no such F, then $f_i(S) = \infty$.)

The main combinatorial property that allows us to solve the problem in polynomial time is that the functions f_i are submodular. We prove something stronger: the functions f_i can be represented in a compact way as the cut function of certain directed graphs.

If D is a directed graph with lengths on the edges and $X \subseteq V(D)$, then $\delta_D(X)$ denotes the total length of the edges leaving X. For $X,Y \subseteq V(D)$, we denote by $\lambda_D(A,B)$ the minimum length of a directed cut that separates A from B, i.e., the minimum of $\delta_D(X)$, taken over all $A \subseteq X \subseteq V(D) \setminus B$ (if $A \cap B \neq \emptyset$, then $\lambda_D(A,B)$ is defined to be ∞).

Definition 1. Let D_i be a directed graph with nonnegative edge lengths. Let s_i and t_i be two distinguished vertices and let A_i be a subset of vertices of D_i. We say that (D_i, s_i, t_i, A_i) *represents* f_i if $f_i(S) = \lambda_{D_i}(S \cup \{s_i\}, (A_i \setminus S) \cup \{t_i\})$ for every $S \subseteq A_i$. If s_i, t_i, A_i are clear from the context, then we simply say that D_i represents f_i.

A function f defined on the subsets of a ground set U is *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ holds for every $X,Y \subseteq U$. For example, it is well known that $\delta_G(X)$ is a submodular function on the subsets of $V(G)$. Submodularity is a powerful unifying concept of combinatorial optimization: classical results on flows, cuts, matchings, and matroids can be considered as consequences of submodularity. The following (quite standard) proposition shows that if a function can be represented in the sense of Definition 1, then the function is submodular. In the proof of Theorem 4, we show that every function f_i can be represented by a directed graph, thus it follows that every f_i is submodular. Although we do not use this observation directly in the paper, it explains in some sense why the problem is polynomial-time solvable.

PROPOSITION 16. *If a function $f_i : 2^{A_i} \to \mathbb{R}^+$ can be represented by (D_i, s_i, t_i, A_i) (in the sense of Definition 1), then f_i is submodular.*

PROOF OF THEOREM 4. We assume that in the given instance of *Steiner forest* each vertex appears only in at most one pair of $\mathcal{D}$. To achieve this, if a vertex v appears in $k > 1$ pairs, then we subdivide an arbitrary edge incident to v by $k-1$ new vertices such that each of the $k-1$ edges on the path formed by v and the new vertices has length 0. Replacing vertex v in a pair by any of the new vertices does not change the problem.

For every $i = 1,\ldots,m$, we compute the values a_i, b_i, and a representation D_i of f_i. In the optimum solution F for the instance $(G_m, \mathcal{D})$, vertices x_m and y_m are either connected or not. Thus the cost of the optimum solutions is the minimum of a_m and $f_m(\emptyset)$ (recall that $A_m = \emptyset$). The value of $f_m(\emptyset)$ can be easily determined by computing the minimum cost s-t cut in D_m.

If G_i is a single edge e, then a_i and b_i are trivial to determine: a_i is the length of e and $b_i = 0$. The directed graph D_i representing f_i

can be obtained from G_i by renaming x_i to s_i, renaming y_i to t_i, and either removing the edge e (if $\mathcal{D}_i = \emptyset$) or replacing e with a directed edge $\overrightarrow{s_i t_i}$ of length ∞ (if $\{u,v\} \in \mathcal{D}_i$).

If G_i is not a single edge, then it is constructed from some G_{j_1} and G_{j_2} either by series or parallel connection. Suppose that a_{j_p}, b_{j_p}, and D_{j_p} for $p = 1,2$ are already known. We show how to compute a_i, b_i, and D_i in this case.

Parallel connection. Suppose that G_i is obtained from G_{j_1} and G_{j_2} by parallel connection. It is easy to see that $a_i = \min\{a_{j_1} + b_{j_2}, b_{j_1} + a_{j_2}\}$ and $b_i = b_{j_1} + b_{j_2}$. To obtain D_i, we join D_{j_1} and D_{j_2} by identifying s_{j_1} with s_{j_2} (call it s_i) and by identifying t_{j_1} with t_{j_2} (call it t_i). Furthermore, for every $\{u,v\} \in \mathcal{D}_i \setminus \{\mathcal{D}_{j_1} \cup \mathcal{D}_{j_2}\}$, we add directed edges $\overrightarrow{uv}$ and $\overrightarrow{vu}$ with length ∞.

To see that D_i represents f_i, suppose that F is the subgraph that realizes the value $f_i(S)$ for some $S \subseteq A_i$. We have to show that there is an appropriate $X \subseteq V(D_i)$ certifying $\lambda_{D_i}(S \cup \{s\}, (A_i \setminus S) \cup \{t\}) \leq \ell(F)$. The graph F is the edge disjoint union of two graphs $F_1 \subseteq G_{j_1}$ and $F_2 \subseteq G_{j_2}$. For $p = 1,2$, let $S^p \subseteq A_{j_p}$ be the set of those vertices that are connected to x_{j_p} in F_p, it is clear that F_p connects $A_{j_p} \setminus S^p$ to y_{j_p}. Since F_p does not connect x_i and y_i, we have that $\ell(F_p) \geq f_{j_p}(S^p)$. Since D_{j_p} represents f_{j_p}, there is a set X_p in D_{j_p} with $S^p \cup \{s_{j_p}\} \subseteq X_p \subseteq V(D_i) \setminus ((A_{j_p} \setminus S^p) \cup \{t_{j_p}\})$, and $\delta_{D_{j_p}}(X_p) = f_{j_p}(S^p)$. We show that $\delta_{D_i}(X_1 \cup X_2) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2)$. Since D_i is obtained from joining D_{j_1} and D_{j_2}, the only thing that has to be verified is that the edges with infinite length added after the join cannot leave $X_1 \cup X_2$. Suppose that there is such an edge $\overrightarrow{uv}$, assume without loss of generality that $u \in X_1$ and $v \in V(D_{j_2}) \setminus X_2$. This means that $u \in S^1$ and $v \notin S^2$. Thus F connects u with x_i and v with y_i, implying that F does not connect u and v. However $\{u,v\} \in \mathcal{D}_i$, a contradiction. Therefore, for the set $X := X_1 \cup X_2$, we have

$$\begin{aligned}\delta_{D_i}(X) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2) &= f_{j_1}(S^1) + f_{j_2}(S^2) \\ &\leq \ell(F_1) + \ell(F_2) = \ell(F) = f_i(S),\end{aligned}$$

proving the existence of the required X.

Suppose now that for some $S \subseteq A_i$, there is a set X with $S \cup \{s_i\} \subseteq X \subseteq V(D_i) \setminus ((A_i \setminus S) \cup \{t_i\})$. We have to show that $\delta_{D_i}(X) \geq f_i(S)$. For $p = 1,2$, let $X_p = X \cap V(D_{j_p})$ and $S^p = A_{j_p} \cap X_p$. Since D_{j_p} represents f_{j_p}, we have that $\delta_{D_{j_p}}(X_p) \geq f_{j_p}(S^p)$. Let F_p be a subgraph of G_{j_p} realizing $f_{j_p}(S^p)$. Let $F = F_1 \cup F_2$; we show that $\ell(F) \geq f_i(A_i)$, since F satisfies all the requirements in the definition of $f_i(A_i)$. It is clear that F does not connect x_i and y_i. Consider a pair $\{u,v\} \in \mathcal{D}_i$. If $\{u,v\} \in \mathcal{D}_{j_p}$, then F connects u and v. Otherwise, let $\{u,v\} \in \mathcal{D}_i \setminus \{\mathcal{D}_{j_1} \cup \mathcal{D}_{j_2}\}$. Suppose that F does not connect u and v, without loss of generality, suppose that $u \in X_1$ and $v \notin X_2$. This means that there is an edge $\overrightarrow{uv}$ of length ∞ in D_i and $\delta_{D_i}(X) = \infty \geq f_i(S)$ follows. Thus we proved $\ell(F) \geq f_i(A_i)$ and we have

$$\begin{aligned}\delta_{D_i}(X) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2) &\geq f_{j_1}(S^1) + f_{j_2}(S^2) \\ &= \ell(F_1) + \ell(F_2) = \ell(F) \geq f_i(S),\end{aligned}$$

what we had to show.

Series connection. Suppose that G_i is obtained from G_{j_1} and G_{j_2} by series connection and let $\mu := y_{j_1} = x_{j_2}$ be the middle vertex. It is easy to see that $a_i = a_{j_1} + a_{j_2}$ (vertex μ has to be connected to both x_i and y_i). To compute b_i, we argue as follows. Denote by $G^R_{j_2}$ the graph obtained from G_{j_2} by swapping the names of distinguished vertices x_{j_2} and y_{j_2}. Observe that the graph G'_i in the definition of b_i arises as the parallel connection of G_{j_1} and $G^R_{j_2}$. It is easy to see that $a^R_{j_2}$, $b^R_{j_2}$, and $f^R_{j_2}$ corresponding to $G^R_{j_2}$ can be defined as $a^R_{j_2} = a_{j_2}$, $b^R_{j_2} = b_{j_2}$, and $f^R_{j_2}(S) = f_{j_2}(A_{j_2} \setminus S)$. Furthermore, if D_{j_2} represents f_{j_2}, then the graph $D^R_{j_2}$ obtained from D_{j_2} by swapping the names of s_{j_2} and t_{j_2} represents $f^R_{j_2}$. Thus we have everything at our disposal to construct a directed graph D'_i that represents the function f'_i corresponding to the parallel connection of G_{j_1} and $G^R_{j_2}$. Now it is easy to see that $b_i = f'_i(A_i)$: graph G'_i is isomorphic to the parallel connection of G_{j_1} and $G^R_{j_2}$ and the definition of b_i requires that A_i is connected to $x_i = y_i$. The value of $f'_i(A_i)$ can be determined by a simple minimum cut computation in D'_i.

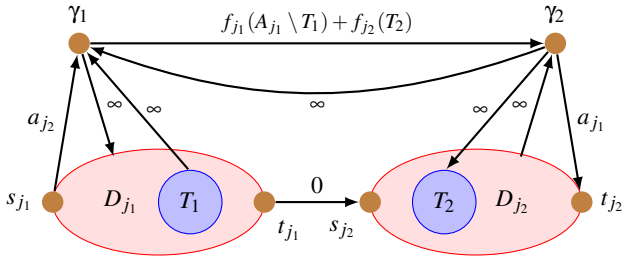

Figure 1: Construction of D_i in a series connection.

Let $T_1 \subseteq A_{j_1}$ contain those vertices v for which there exists a pair $\{v,u\} \in \mathcal{D}_i$ with $u \in A_{j_2}$ and let $T_2 \subseteq A_{j_2}$ contain those vertices v for which there exists a pair $\{v,u\} \in \mathcal{D}_i$ with $u \in A_{j_1}$. Observe that $A_i = (A_{j_1} \setminus T_1) \cup (A_{j_2} \setminus T_2)$ (here we are using the fact that each vertex is contained in at most one pair, thus a $v \in T_1$ cannot be part of any pair $\{v,u\}$ with $u \notin V(D_i)$). To construct D_i, we connect D_{j_1} and D_{j_2} with an edge $\overrightarrow{t_{j_1} s_{j_2}}$ of length 0 and set $s_i := s_{j_1}$ and $t_i := t_{j_2}$. Furthermore, we introduce two new vertices γ_1, γ_2 and add the following edges (see Figure 1):

- $\overrightarrow{s_{j_1}\gamma_1}$ with length a_{j_2},
- $\overrightarrow{\gamma_1\gamma_2}$ with length $f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2)$,
- $\overrightarrow{\gamma_2 t_{j_2}}$ with length a_{j_1},
- $\overrightarrow{\gamma_2\gamma_1}$ with length ∞,
- $\overrightarrow{\gamma_1 v}$ with length ∞ for every $v \in V(D_{j_1})$,
- $\overrightarrow{v\gamma_2}$ with length ∞ for every $v \in V(D_{j_2})$,
- $\overrightarrow{v\gamma_1}$ with length ∞ for every $v \in T_1$, and
- $\overrightarrow{\gamma_2 v}$ with length ∞ for every $v \in T_2$.

Suppose that F is the subgraph that realizes the value $f_i(S)$ for some $S \subseteq A_i$; we have to show that D_i has an appropriate cut with value $f_i(S)$. Subgraph F is the edge-disjoint union of subgraphs $F_1 \subseteq G_{j_1}$ and $F_2 \subseteq G_{j_2}$. We consider 3 cases: in subgraph F, vertex μ is either connected to neither x_{j_1} nor y_{j_2}, connected only to x_{j_1}, or connected only to y_{j_2}.

Case 1: μ is connected to neither x_{j_1} nor y_{j_2}. In this case, vertices of A_{j_1} are not connected to y_i and vertices of A_{j_2} are not connected to x_i, hence $S = A_i \cap A_{j_1} = A_{j_1} \setminus T_1$ is the only possibility. Furthermore, F connects both T_1 and T_2 to μ. It follows that $\ell(F) = \ell(F_1) + \ell(F_2) \geq f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2)$. Set $X = V(D_i) \cup \{\gamma_1\}$: now we have $\delta_{D_i}(X) = f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2) \leq \ell(F)$, X contains $(A_{j_1} \setminus T_1) \cup \{s_i\}$, and is disjoint from $(A_{j_2} \setminus T_2) \cup \{t_i\}$.

Case 2: μ is connected only to x_i. This is only possible if $A_{j_1} \setminus T_1 \subseteq S$. Clearly, $\ell(F_1) \geq a_{j_1}$. Subgraph F_2 has to connect every vertex in $(S \cap A_{j_2}) \cup T_2$ to x_{j_2} and every vertex $A_i \setminus S = A_{j_2} \setminus (S \cup T_2)$ to y_{j_2}. Hence $\ell(F_2) \geq f_{j_2}((S \cap A_{j_2}) \cup T_2)$ and let $X_2 \subseteq V(D_{j_2})$ be the corresponding cut in D_{j_2}. Set $X := X_2 \cup V(D_{j_1}) \cup \{\gamma_1, \gamma_2\}$, we have $\delta_{D_i}(X) = f_{j_2}((S \cap A_{j_2}) \cup T_2) + a_{j_1} \leq \ell(F_2) + a_{j_1} \leq \ell(F)$ (note that no edge with infinite length leaves X since $T_2 \subseteq X_2$). As X contains S and contains none of the vertices in $A_i \setminus S$, we proved the existence of the required cut.

Case 3: μ is connected only to y_i. Similar to case 2.

Suppose now that for some $S \subseteq A_i$, there is a set $X \subseteq V(D_i)$ with $S \cup \{s_i\} \subseteq X \subseteq V(D_i) \setminus ((A_i \setminus S) \cup \{t_i\})$. We have to show that $\delta_{D_i}(X) \geq f_i(S)$. If $\delta_{D_i}(X) = \infty$, then there is nothing to show. In particular, because of the edge $\overrightarrow{\gamma_2\gamma_1}$, we are trivially done if $\gamma_2 \in X$ and $\gamma_1 \notin X$. Thus we have to consider only 3 cases depending which of γ_1, γ_2 are contained in X.

Case 1: $\gamma_1 \in X$, $\gamma_2 \notin X$. In this case, the edges having length ∞ ensure that $V(D_{j_1}) \subseteq X$ and $V(D_{j_2}) \cap X = \emptyset$, thus $\delta_{D_i}(X) = \ell(\overrightarrow{\gamma_1\gamma_2}) + \ell(\overrightarrow{t_{j_1}s_{j_2}}) = f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_{j_2})$. We also have $S = A_{j_1} \setminus T_1$. Now it is easy to see that $f_i(S) \leq f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_{j_2})$: taking the union of some F_1 realizing $f_{j_1}(A_{j_1} \setminus T_1)$ and some F_2 realizing $f_{j_2}(T_{j_2})$, we get a subset F realizing $f_i(S)$.

Case 2: $\gamma_1, \gamma_2 \in X$. The edges starting from γ_1 and having length ∞ ensure that $V(D_{j_1}) \subseteq X$. Furthermore, $\gamma_2 \in X$ ensures that $T_2 \subseteq X$. Let $X_2 := X \cap V(D_{j_2})$, we have $X_2 \cap A_{j_2} = T_2 \cup (S \cap A_{j_2})$, which implies $\delta_{D_{j_2}}(X_2) \geq f_{j_2}(T_2 \cup (S \cap A_{j_2}))$. Observe that $\delta_{D_i}(X) = a_{j_1} + \delta_{D_{j_2}}(X_2)$ (the term a_{j_1} comes from the edge $\overrightarrow{\gamma_2 t_i}$). Let F_1 be a subset of G_{j_1} realizing a_{j_1} and let F_2 be a subset of G_{j_2} realizing $f_{j_2}(T_2 \cup (S \cap A_{j_2}))$. Let $F := F_1 \cup F_2$, note that F connects vertices S with x_i, vertices $A_i \setminus S$ with y_i, and vertices in $T_1 \cup T_2$ with μ. Thus $f_i(S) \leq \ell(F) = a_{j_1} + f_{j_2}(T_2 \cup (S \cap A_{j_2})) = \delta_{D_i}(X)$, what we had to show.

Case 3: $\gamma_1, \gamma_2 \notin X$. Similar to Case 2. □

7. REFERENCES

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.

[2] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *STOC*, pages 134–144, 1991.

[3] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. In *FOCS*, 2009.

[4] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *FOCS*, page 2, 1996.

[5] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[6] M. Bateni, M. Hajiaghayi, and D. Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *CoRR*, abs/0911.5143, 2009.

[7] P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. In *SODA*, pages 325–334, 1992.

[8] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.

[9] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[10] G. Borradaile, E. D. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded genus graphs. In *STACS*, pages 171–182, 2009.

[11] G. Borradaile, P. N. Klein, and C. Mathieu. A polynomial-time approximation scheme for Euclidean Steiner forest. In *FOCS*, pages 115–124, 2008.

[12] G. Borradaile, C. Mathieu, and P. N. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *SODA*, pages 1285–1294, 2007.

[13] E. D. Demaine, M. Hajiaghayi, and B. Mohar. Approximation algorithms via contraction decomposition. In *SODA*, pages 278–287, 2007.

[14] R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[15] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[16] E. Gassner. The Steiner forest problem revisited. *J. Disc. Alg.*, In Press, Corrected Proof, 2009.

[17] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[18] S. Hougardy and H. J. Prömel. A 1.598 approximation algorithm for the Steiner problem in graphs. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 448–453, 1999.

[19] R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[20] M. Karpinski and A. Zelikovsky. New approximation algorithms for the Steiner tree problems. *J. Comb. Opt.*, 1(1):47–65, 1997.

[21] P. N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM J. Comput.*, 37(6):1926–1952, 2008.

[22] D. Marx. NP-completeness of list coloring and precoloring extension on the edges of planar graphs. *J. Graph Theory*, 49(4):313–324, 2005.

[23] D. Marx. Complexity results for minimum sum edge coloring. *Discrete Applied Mathematics*, 157(5):1034–1045, 2009.

[24] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[25] H. J. Prömel and A. Steger. RNC-approximation algorithms for the Steiner problem. In *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 559–570, 1997.

[26] M. B. Richey and R. G. Parker. On multiple Steiner subgraph problems. *Networks*, 16(4):423–438, 1986.

[27] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM J. Disc. Math.*, 19(1):122–134, 2005.

[28] M. Thimm. On the approximability of the Steiner tree problem. *Theor. Comput. Sci.*, 295(1-3):387–402, 2003.

[29] A. Zelikovsky. An 11/6-approximation for the Steiner problem on graphs. In *Fourth Czechoslovakian Symp. Combinatorics, Graphs, and Complexity (1990) in Annals of Discrete Mathematics*, volume 51, pages 351–354, 1992.

[30] A. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[31] A. Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. Technical report, University of Virginia, Charlottesville, VA, USA, 1996.

[32] X. Zhou and T. Nishizeki. The edge-disjoint paths problem is NP-complete for partial k-trees. In *Algorithms and computation (Taejon, 1998)*, pages 417–426. Springer, Berlin, 1998.