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

MOHAMMADHOSSEIN BATENI
Princeton University
MOHAMMADTAGHI HAJIAGHAYI
University of Maryland at College Park
DÁNIEL MARX
Humboldt-Universität zu Berlin

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.1.0 [**NUMERICAL ANALYSIS**]: General; G.2.2 [**DISCRETE MATHEMATICS**]: Graph Theory—*Network problems*

General Terms: Algorithms, Design, Performance, Theory

Additional Key Words and Phrases: Approximation algorithm, bounded-treewidth, dynamic programming, network design, planar graph, PTAS, series-parallel graph, Steiner forest

---

M. Bateni, Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540. The author is also with the Center for Computational Intractability at Princeton; Email: mbateni@cs.princeton.edu. 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, and a Charlotte Elizabeth Procter fellowship.

M. Hajiaghayi, Department of Computer Science, University of Maryland at College Park, 115 A.V. Williams Building, College Park, MD 20742; Email: hajiagha@cs.umd.edu. Supported in part by NSF CAREER award 1053605 and Google Faculty Reseach Award. The author is also with AT&T Labs - Research.

D. Marx, Institut für Informatik, Humboldt Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany; Email: dmarx@cs.bme.hu. Research supported in part by ERC Advanced grant DMMCA, the Alexander von Humboldt Foundation, and the Hungarian National Research Fund (Grant Number OTKA 67651).

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-length 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 [1995] (see also Goemans and Williamson [1995]). Since the conference version of the Agrawal, Klein and Ravi [1991] appeared, there have been no improved approximation algorithms invented for Steiner forest. Recently Borradaile, Klein and Mathieu [2008] 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 PTASes in planar graphs—we mention some of the follow-up works, especially those using this technique, at the end of this section.

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 [1975]. The problem remains hard even on planar graphs [Garey and Johnson 1977]. In contrast to Steiner forest, a long sequence of papers give approximation factors better than 2 for this problem [Zelikovsky 1992; 1993; Berman and Ramaiyer 1992; Zelikovsky 1996; Prömel and Steger 1997; Karpinski and Zelikovsky 1997; Hougardy and Prömel 1999; Robins and Zelikovsky 2005; Byrka et al. 2010]; the current best approximation ratio is 1.39 [Byrka et al. 2010]. Since the problem is APX-hard in general graphs [Bern and Plassmann 1989; Thimm 2003], we do not expect to obtain a PTAS for this problem in general graphs. However, for the Euclidean Steiner tree problem, the classic works of Arora [1998] and Mitchell [1999] 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 [1996]. Only in 2007, Borradaile et al. [2009] settle this problem with a nice technique of constructing *light spanners* for Steiner 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, the ideas in [Baker 1994] and the reformulations in [Demaine et al. 2007; Klein 2008] provide a general method of reducing many optimization problems on planar (and bounded-genus) graphs to bounded-treewidth graphs. In particular, a keystone blackbox in the algorithm of Borradaile et al. [2009] 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 [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 [Gassner 2009]. 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 1.1. *For any constant $\bar{\epsilon} > 0$, there is a polynomial-time $(1 + \bar{\epsilon})$-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 length is no more than a constant factor times the length of the optimal Steiner forest, and furthermore, it contains a nearly optimal Steiner forest. Denote by $\mathrm{OPT}_{\mathcal{D}}(G)$ the minimum length of a Steiner forest of G satisfying (connecting) all the demands in $\mathcal{D}$. We sometimes use OPT instead of $\mathrm{OPT}_{\mathcal{D}}(G)$ when $\mathcal{D}$ and G are easily inferred from the context. 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 $(1 + \epsilon)OPT_{\mathcal{D}}(G)$, namely, $\mathrm{OPT}_{\mathcal{D}}(H) \leq (1 + \epsilon)\mathrm{OPT}_{\mathcal{D}}(G)$.

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

THEOREM 1.2. *Given any fixed $\epsilon > 0$, 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 1.2 heavily relies on a novel clustering method presented in Theorem 1.3 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. [2009], 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 1.3. *Given an $\epsilon > 0$, 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 lengths of all the trees T_i is no more than $(\frac{4}{\epsilon} + 2)\mathrm{OPT}_{\mathcal{D}}(G_{in})$.*
(4) *The sum of the lengths of minimum Steiner forests of all demand sets $\mathcal{D}_i$ is no more than $1 + \epsilon$ times the length of a minimum Steiner forest of G_{in}; i.e., $\sum_i \mathrm{OPT}_{\mathcal{D}_i}(G_{in}) \leq (1 + \epsilon)\mathrm{OPT}_{\mathcal{D}}(G_{in})$.*

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 1.3 in Section 3. Roughly speaking, the algorithm here first identifies 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 so 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. Then 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 et al. 1991; Goemans and Williamson 1995]. Indeed our analysis strengthens these previous approaches by proving certain 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 [2009] who improve the best approximation factor for prize-collecting Steiner tree to $2 - \epsilon$, for some constant $\epsilon > 0$. 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 paint (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 the super vertices to each other with a length 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 optimal 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 (especially to obtain PTASes).

To obtain a PTAS as promised in Theorem 1.1, we first construct a spanner Steiner forest based on Theorem 1.2. On this spanner, we utilize a technique due to [Klein 2008; Demaine et al. 2007] that reduces the problem of obtaining a PTAS in a planar (and more generally, bounded-genus) graph whose total edge length is within a constant factor of the optimal solution to that of finding an optimal solution in a graph of bounded treewidth. (See the proof of Theorem 1.1 in Section 5 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, too.

Resolving an open question of Richey and Parker [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 [Gassner 2009].

THEOREM 1.4. *The Steiner forest problem can be solved in polynomial time for subgraphs of series-parallel graphs (i.e., graphs of treewidth at most 2).*

A series-parallel graph can be built form elementary blocks using two operations: parallel connection and series connection. The algorithm of Theorem 1.4 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 optimal solution by the computation of a minimum cut.

Surprisingly, it turns out that the problem becomes NP-hard on graphs of treewidth at most 3 [Gassner 2009]. For completeness, in Section 8 we give a (different) NP-hardness proof, that highlights how submodularity (and hence the approach of Theorem 1.4) 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 [Zhou and Nishizeki 1998; Marx 2005; 2009; Richey and Parker 1986; Bateni et al. 2011], 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 2008; Demaine et al. 2007] can still be applied when we have a PTAS for graphs of bounded treewidth. In Section 6, we demonstrate such a PTAS.

THEOREM 1.5. *For every fixed $w \geq 1$ and $\epsilon > 0$, there is a polynomial-time $(1+\epsilon)$-approximation algorithm for Steiner Forest on graphs with treewidth at most w.*

Note that the exponent of the polynomial in Theorem 1.5 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 1.5 is to reduce the set of partitions considered in the dynamic programming algorithm to a polynomially bounded subset, in a way that an $(1 + \epsilon)$-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 $(1 + \epsilon)$-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 length, 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 length. 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.

1.2 Follow-up work

The prize-collecting clustering technique was first implicitly used in Archer et al. [2009] for obtaining an approximation ratio of prize-collecting Steiner tree and prize-collecting traveling salesman problems to a constant below two. The technique was then formalized in the conference version of the current work and led to a PTAS for planar Steiner forest. Bateni, Chekuri, Ene, Hajiaghayi, Korula, and Marx [2011] subsequently generalized the technique and provided reductions from planar graphs to bounded-treewidth graphs for a large class of prize-collecting Steiner network problems. This resulted in PTASes for prize-collecting Steiner tree, prize-collecting TSP and prize-collecting stroll on planar graphs. In another work, the technique has found application in presenting a PTAS for the planar multiway cut problem [Bateni, Hajiaghayi, Klein, and Mathieu 2011].

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 $\mathrm{dist}_G(x,y)$. For an edge set E, we denote by $\ell(E) := \sum_{e\in E} \ell(e)$ the total length of edges in E, where $\ell(e)$ denotes the length of edge e in G.

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 all but the shortest among parallel edges. More formally, the contracted graph G/e is formed by the replacement of u and v with a single vertex such that edges incident to the new vertex are the edges other than e that were incident with u or v. To obtain a simple graph, we first remove all self-loops in the resulting graph. In case of multiple edges, we only keep the shortest edge and remove all the rest. The contraction G/E' is defined as the result of iteratively contracting all the edges of E' in G, i.e., $G/E' := G/e_1/e_2/\dots/e_k$ if $E' = \{e_1, e_2, \dots, e_k\}$. Clearly, the planarity of G is preserved after the contraction. Similarly, contracting edges does not increase the length of an optimal Steiner forest.

The boundary of a face of a planar embedded graph is the set of edges adjacent to the face; it does not always form a simple cycle. The boundary ∂H of a planar embedded graph H is the set of edges bounding the infinite face. An edge is strictly enclosed by the boundary of H if the edge belongs to H but not to ∂H.

Now we define the basic notion of treewidth, as introduced by [Robertson and Seymour 1986]. To define this notion, we consider representing a graph by a tree structure, called a tree decomposition. More precisely, a *tree decomposition* of a graph $G(V,E)$ is a pair $(T,\mathcal{B})$ in which $T(I,F)$ is a tree and $\mathcal{B} = \{B_i \mid i \in I\}$ is a family of subsets of $V(G)$ such that

(1) $\bigcup_{i\in I} B_i = V$;
(2) for each edge $e = (u,v) \in E$, there exists an $i \in I$ such that both u and v belong to B_i; and
(3) the set of nodes $\{i \in I \,|\, v \in B_i\}$ forms a connected subtree of T for every $v \in V$.

To distinguish between vertices of the original graph G and vertices of T in the tree decomposition, we call vertices of T *nodes* and their corresponding B_i's *bags*. The *width* of the tree decomposition is the maximum size of a bag in $\mathcal{B}$ minus 1. The *treewidth* of a graph G, denoted $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G.

For algorithmic purposes, it is convenient to define a restricted form of tree decomposition. We say that a tree decomposition $(T, \mathcal{B})$ is *nice* if the tree T is a rooted tree such that for every $i \in I$ either

(1) i has no children (i is a *leaf node*),
(2) i has exactly two children i_1, i_2 and $B_i = B_{i_1} = B_{i_2}$ holds (i is a *join node*),
(3) i has a single child i' and $B_i = B_{i'} \cup \{v\}$ for some $v \in V$ (i is an *introduce node*), or
(4) i has a single child i' and $B_i = B_{i'} \setminus \{v\}$ for some $v \in V$ (i is a *forget node*).

It is well-known that every tree decomposition can be transformed into a nice tree decomposition of the same width in polynomial time. Furthermore, we can assume that the root bag contains only a single vertex.

We will use the following lemma to obtain a nice tree decomposition with some further properties (a related trick was used in [Marx 2007], the proof is similar):

LEMMA 2.1. *Let G be a graph having no adjacent degree 1 vertices. G has a nice tree decomposition of polynomial size having the following two additional properties:*

(*1*) *No introduce node introduces a degree 1 vertex.*
(*2*) *The vertices in a join node have degree greater than 1.*

PROOF. Consider a nice tree decomposition of graph G. First, if v is a vertex of degree 1, then we can assume that v appears only in one bag: if w is the unique neighbor of v, then it is sufficient that v appears in any one of the bags that contain w. Let $B_v = \{v, x_1, \ldots, x_t\}$ be this bag where $x_1 = w$. We modify the tree decomposition as follows. We replace B_v with $B'_v = B_v \setminus \{v\}$, insert a bag $B''_v = B_v \setminus \{v\}$ between B'_v and its parent, and create a new bag $B^t = B_v \setminus \{v\}$ that is the other child of B''_v (thus B''_v is a join node). For $i = 1, \ldots, t-1$, let $B^i = \{x_1, \ldots, x_i\}$, and let B^i be the child of B^{i+1}. Finally, let $B_w = \{w, v\}$ be the child of B^1 and let $B = \{v\}$ be the child of B_w. Observe that B^i $(2 \leq i \leq t)$, B_w are introduce nodes, B^1 is a forget node, and B is a leaf node. This operation ensures that vertex v appears only in a leaf node. It is clear that after repeating this operation for every vertex of degree 1, the two required properties will hold. $\square$

We also need a basic notion of embedding; see, e.g., [Robertson and Seymour 1994; Cabello and Mohar 2005]. In this paper, an *embedding* refers to a 2-*cell embedding*, i.e., a drawing of the vertices and edges of the graph as points and arcs in a surface such that every face (connected component obtained after removing edges and vertices of the embedded graph) is homeomorphic to an open disk. We use basic terminology and notions about embeddings as introduced in [Mohar and Thomassen 2001]. We only consider compact surfaces without boundary. Occasionally, we refer to embeddings in the plane, when we actually mean embeddings in the 2-sphere. If S is a surface, then for a graph G that is (2-cell) embedded in S with f facial walks, the number $g = 2 - |V(G)| + |E(G)| - f$ is independent of G and is called the *Euler genus* of S. The Euler genus coincides with the crosscap number if S is non-orientable, and equals twice the usual genus if the surface S is orientable.

3. PRIZE-COLLECTING CLUSTERING

The goal of this section is to prove Theorem 1.3, that allows us to break up the input Steiner forest instance into simpler ones. The proof relies on the prize-collecting clustering technique, that produces a partition of graph vertices corresponding to potential values assigned to them. Roughly speaking, vertices close to each other (relative to their potentials) are placed in one cluster, and those far from each other will be part of different clusters.

It is more instructive to state the prize-collecting theorem in abstract terms, and then use it to prove the particular theorem needed for Steiner forest. This abstraction cleans up the proofs.

THEOREM 3.1 (PRIZE-COLLECTING CLUSTERING). *Let $G(V, E)$ be a graph with a nonnegative edge length $\ell(e)$ for each edge e and a potential ϕ_v for each vertex v. In polynomial time, we can find a subgraph Z such that*

(*1*) *the length of Z is at most $2\sum_{v\in V}\phi_v$, and*

(*2*) *for any subgraph L of G, there is a set Q of vertices such that*
  - (*a*) *$\sum_{v\in Q}\phi_v$ is at most the length of L, and*
  - (*b*) *if two vertices $v_1, v_2 \notin Q$ are connected by L, then they are in the same component of Z.*

The theorem is proved by presenting and analyzing a procedure, `PC-Clustering`, which takes as input G, ℓ, ϕ, and outputs Z; however, this procedure does not compute Q since such a computation requires the knowledge of L. In fact, Q need not even be computable (in polynomial time) from L—although it is possible—and the very guarantee that a suitable Q exists for each L is sufficient for the applications of the theorem.

Before giving the proof of Theorem 3.1 in Section 3.1, we argue that it implies Theorem 1.3. The procedure for doing so is shown in Algorithm 1 (`PC-Partition`).

Algorithm 1 `PC-Partition`

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

```
1: Use the algorithm of [Goemans and Williamson 1995] 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 $\phi_v$ be $\frac{1}{\epsilon}$ times the length of the tree $T_i^*$ corresponding to $v$,
   and zero if there is no such tree.
4: Let $F_2$ be the output of PC-Clustering invoked on $G$ and $\phi_v$.
5: Construct $F$ from $F_2$ by uncontracting all the trees $T_i^*$.
6: Let $F$ consist of tree components $T_i$.
7: Output the set of trees $\{T_i\}$, along with $\mathcal{D}_i := \{(s,t) \in \mathcal{D} : s,t \in V(T_i)\}$.
```

PROOF OF THEOREM 1.3. We start with a 2-approximate solution F^* satisfying all the demands in $\mathcal{D}$ (such a solution can be found via Goemans-Williamson's Steiner forest algorithm, for instance). In the following, we extend F^* by connecting some of its components to make the trees T_i. It is easy to see that this construction guarantees the first two conditions of Theorem 1.3. 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 $\frac{1}{\epsilon}$ times the length of the tree component of F^* corresponding to v in case v is the contraction of a tree component, and zero otherwise.

Let Z be the subgraph of G given by Theorem 3.1. Let Z_{in} be the subgraph of G_{in} obtained from Z by uncontracting the components of F^* and adding F^* to Z_{in}; as F^* is a solution, Z_{in} is a solution as well. Let $T_1, \ldots, T_k$ be spanning trees of the components of Z_{in} and let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be the set of demands spanned by these trees. It is clear that the first two conditions of the theorem hold. The length of Z_{in} is the length of F^* (which is at most 2OPT) plus the length of Z (which is at most $2\sum_{v\in V}\phi_v \le \frac{4}{\epsilon}\text{OPT}$), giving the third condition.

Let OPT be an optimal solution and let L be the corresponding subgraph of G (obtained by contracting the components of F^*). Let Q be the set of vertices of G given by the second condition of Theorem 3.1. For every node in Q, there is a corresponding component of F^*; let Q_{in} be the forest of G_{in} composed from all these components. From the way the potential ϕ_v was defined, we have $\ell(Q_{in}) = \epsilon\sum_{v\in Q}\phi_v \le \epsilon\ell(L) \le \epsilon\ell(\text{OPT})$, where in the second inequality, we used condition 2(a) of Theorem 3.1.

To show that the last condition holds, for every $\mathcal{D}_i$ we construct a subgraph H_i that satisfies the demands in $\mathcal{D}_i$. For every demand in $\mathcal{D}_i$, if the component K of F^* satisfying the demand belongs to Q_{in}, then we put K into H_i; otherwise, we put the component of OPT satisfying the demand into H_i. Observe that each component of Q_{in} is used in at most one of the H_i's: as Q_{in} is a subgraph of Z_{in}, all the demands satisfied by a component K of Q_{in} belong to the same $\mathcal{D}_i$. Furthermore, we claim that each component of OPT is used in at most one of the H_i's. Suppose that a component K of OPT was used in both H_i and H_j, i.e., K satisfies a demand in $\mathcal{D}_i$ and a demand in $\mathcal{D}_j$. The components of F^* satisfying these two demands are *not* in Q_{in} (otherwise we would have put these components into H_i or H_j instead of K), thus they correspond to nodes $v_1, v_2 \notin Q$ in the contracted graph G. Thus L, the contracted version of OPT, connects two nodes $v_1, v_2 \notin Q$. Condition 2(b) of Theorem 3.1 implies that v_1 and v_2 are in the same component of Z and hence the two demands are satisfied by the same component of Z_{in}. This contradicts that the two demands are in two different sets $\mathcal{D}_i$ and $\mathcal{D}_j$.

Since every component of Q_{in} and every component of OPT is used by at most one of the H_i's, we have $\sum_{i=1}^{k}\ell(H_i) \le \ell(OPT) + \ell(Q_{in}) \le (1+\epsilon)\ell(OPT)$. This establishes the last condition of the theorem. $\square$

3.1 Proof of Theorem 3.1

In this section, we describe an algorithm `PC-Clustering` that is used to prove Theorem 3.1. The algorithm as well as its analysis bears similarities to the primal-dual method due to [Agrawal et al. 1995; Goemans and Williamson 1995]. 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).

The following system of linear equations is central to our algorithm. The variable $y_{S,v}$ is defined for every v and S with $v \in S \subseteq V$, and $\ell(e)$ denotes the length of the edge e.

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

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

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

These constraints are quite 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.

Although there are exponentially many variables here, only a polynomial number of them will take nonzero values during the algorithm, and the algorithm runs in polynomial time.

The solution is built up in two stages. First we perform an *unrooted growth* to find a forest F_1 and a corresponding vector y. (During the process, we maintain a vector y satisfying all these constraints, and at the end, all the constraints (2) hold with equality.) In the second stage, we *prune* some of the edges of F_1 to get another forest F_2. Below we describe the two phases of Algorithm 2 (`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 paints 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 paint 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

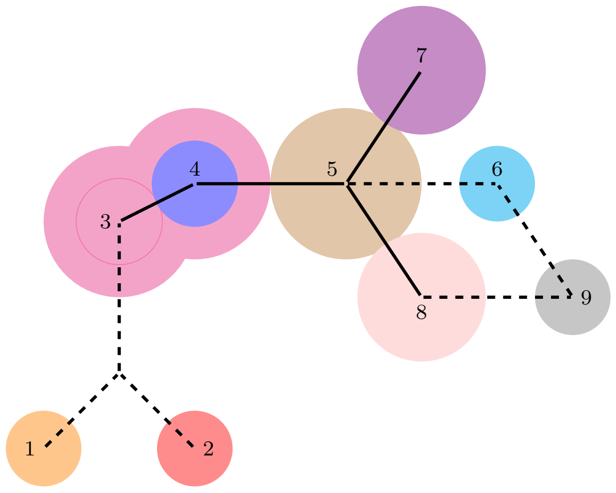

Fig. 1. `PC-Clustering` paints the graph with different colors (corresponding to potentials of vertices), and each part of an edge may get a different color. Solid edges represent edges bought by `PC-Clustering`, whose result gives rise to five connected components, namely $K_1 = \{1\}, K_2 = \{2\}, K_3 = \{3, 4, 5, 7, 8\}, K_4 = \{6\}$ and $K_5 = \{9\}$. (The final pruning step may erase some of the edges and break up certain components, but for simplicity of presentation in this figure, we assume these are the actual components.) The dashed edges correspond to another solution, say the optimum. The "length" of two components connected by such a solution can be charged to the length of this solution. For instance, the path from 1 to 3 passes through the colors corresponding to 1 and 3, and, since both nodes are exhausted, the length of the path is at least the sum of the potentials of the two nodes—in general, at least one of the involved nodes is exhausted by the solution whenever it connects two separate components of `PC-Clustering`. Thus, we can leave these two nodes out of consideration, and continue this process. As a result, we can solve demands in each of the remaining components separately, and only pay a small additional length (due to charging).

the portions of color v are consecutive on an edge.[1] 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.

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.

[1]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.

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, $\delta(C) \cap F_2 = \emptyset$. We argue that a pruned cluster cannot have non-empty and proper intersection with a connected component of F_2. Notice that, at the time the cluster C is pruned, the single remaining edge of $F_2 \cap \delta(C)$ is removed from F_2, thus the final F_2 cannot have any edge of $\delta(C)$. Therefore, no connected component of F_2 can have two vertices such that one is inside C and the other is not.

Before, we can give the algorithm, we need to define $\mathcal{C}(v)$ as the cluster currently containing the vertex $v \in V'$, i.e., $\mathcal{C}(v) := C$ for any $v \in C \in \mathcal{C}$.

Algorithm 2 `PC-Clustering`

```
Input: graph $G(V, E)$, and potentials $\phi_v \geq 0$.
Output: forest $F_2$.
1: Let $F_1 \leftarrow \emptyset$.
2: Let $y_{S,v} \leftarrow 0$ for any $v \in S \subseteq V$.
3: Let $\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V\}$.
4: while there is a live vertex do
5:     Let $\eta$ be the largest possible value such that simultaneously increasing $y_C$ by
       $\eta$ for all active clusters $C$ does not violate Constraints (1)-(3).
6:     Let $y_{\mathcal{C}(v),v} \leftarrow y_{\mathcal{C}(v),v} + \frac{\eta}{\kappa(\mathcal{C}(v))}$ for all live vertices $v$.
7:     if $\exists e \in E$ that is tight and connects two clusters then
8:         Pick one such edge $e = (u, v)$.
9:         Let $F_1 \leftarrow F_1 \cup \{e\}$.
10:        Let $C \leftarrow \mathcal{C}(u) \cup \mathcal{C}(v)$.
11:        Let $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{\mathcal{C}(u), \mathcal{C}(v)\}$.
12:        Let $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$.
13: Let $F_2 \leftarrow F_1$.
14: Let $\mathcal{B} \leftarrow \{S \in \mathcal{S} | \sum_{S' \subseteq S} \sum_{v \in S'} y_{S',v} = \sum_{v \in S} \phi_v\}$.
15: while $\exists S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$ for an edge $e$ do
16:     Let $F_2 \leftarrow F_2 \setminus \{e\}$.
17: Output $F_2$.
```

Notice that Line 4 of `PC-Clustering` ensures that, at the end of the algorithm, all budgets are exhausted because no vertex is alive.

Observation 3.2. *When `PC-Clustering` terminates, Inequalities* (2) *hold with equality.*

We first bound the length of the forest F_2. The following lemma is similar to the analysis of the algorithm in [Goemans and Williamson 1995]. However, we do not have a primal LP to give a bound on the dual. Rather, the upper bound for the length is the sum of all the potential values $\sum_v \phi_v$. In addition, we bound the length of a forest F_2 that may have more than one connected component, whereas the prize-collecting Steiner tree algorithm of [Goemans and Williamson 1995] finds a connected graph at the end.

Lemma 3.3. *The length of F_2 is at most $2\sum_{v\in V}\phi_v$.*

`PC-Clustering` has a notion of time, and the algorithm is defined by how the forest F_1, F_2 and dual variables $y_{S,v}$ grow over time. Conceptually, it is helpful to think of `PC-Clustering` as progressing continuously over time, although the actual computation is confined to a sequence of discrete *event points*. Time begins at 0 and unfolds in *epochs*, which are the intervals of time between two consecutive event points (possibly an empty interval, if two event points occur simultaneously). There are two types of event points, *tight edge* and *cluster death* events, which we described previously. Notice that, during each epoch, each cluster is either active or inactive, and each active cluster C increases its y_C value at rate 1 for the duration of the epoch, while all other duals remain unchanged. At time 0, the (singleton) clusters with strictly positive penalty are active.

Proof of Lemma 3.3. The strategy is to prove that the length of the forest F_2 is at most $2\sum_{v\in S\subseteq V} y_{S,v} \le 2\sum_{v\in V}\phi_v$. The inequality 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 length $\ell(e)$ 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, $\ell(e) = \sum_j \sum_{C:e\in\delta(C)} y_C^{(j)}$. We will now prove that the total edge length 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 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 $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|) - 2|\mathcal{C}_{dead}| = 2|\mathcal{C}_{act}|$ 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 3.4. *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. As C cuts e, it only has one endpoint of the edge. Then, the laminarity of the clusters $\mathcal{S}$ gives $C \subset S, S'$. In addition, we show that C contains exactly one endpoint of the path p (as opposed to exactly one endpoint of the edge e). This holds because, if C contained neither or both endpoints of p, the cluster C could not cut p at exactly one edge. Call the endpoint of p in this cluster 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 when 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 painted 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 3.6, we are going to use potentials as a lower bound on the length of a graph. (When it is invoked in Theorem 1.3, this serves as a lower bound on the optimum.) 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 with $y_{S,u} > 0$.

Lemma 3.5. *If a subgraph $H(V, E')$ of G connects two vertices u, v from different components of F_2, then H exhausts the color corresponding to at least one of u and v.*

Proof. Suppose that H exhausts neither u nor v: there is a set S containing u and a set S' containing v such that $y_{S,v}, y_{S',u} > 0$ and $E' \cap \delta(S) = E' \cap \delta(S') = \emptyset$. Since H connects u and v, this is only possible if u and v are both in S and S'. By Lemma 3.4, this implies that F_2 connects u and v, a contradiction. □

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

LEMMA 3.6. *Let Q be the set of colors exhausted by subgraph G' of G. The length of $G'(V, E')$ is at least $\sum_{v \in Q} \phi_v$.*

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 length of G' can be charged to $y_{S,v}$. Hence, the total length of the graph G' is at least as large as the total amount of colors paid for by Q. We now provide a formal proof.

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

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

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

$$= \sum_{v \in Q} \phi_v \qquad \text{by Observation 3.2.} \qquad \square$$

Now we are ready to prove Theorem 3.1.

PROOF OF THEOREM 3.1. The subgraph Z is the forest F_2. Condition 1 is given by Lemma 3.3. For condition 2, let Q be the set of vertices exhausted by L. Now conditions 2(a) and 2(b) are proved by Lemma 3.6 and Lemma 3.5, respectively. $\square$

4. CONSTRUCTING THE SPANNER

The goal of this section is to prove Theorem 1.2. Recall that we are given a graph $G_{in}(V_{in}, E_{in})$, and a set of demands $\mathcal{D}$. From Theorem 1.3, 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 terminals Q_i in demand set $\mathcal{D}_i$, and the total length of trees T_i is $O(\text{OPT})$. The construction goes along the same lines as those of [Borradaile, Demaine, and Tazari 2009; Borradaile, Klein, and Mathieu 2009], yet there are certain differences in the analysis. The construction is carried out in three steps. We separately build a graph H_i for each T_i, and finally let H be the union of all graphs H_i.

Borradaile, Klein and Mathieu [2007; 2009] developed a PTAS for the Steiner tree problem in planar graphs; this technique was later extended to bounded-genus graphs [Borradaile et al. 2009]. The method involves finding a grid-like subgraph called the *mortar graph* that spans the input terminals and has length $O(\mathrm{OPT})$. Each face of the mortar graph corresponds to a subgraph of the original graph, called a brick. The set of feasible Steiner trees is restricted to those that cross brick boundaries only at a small number (per brick) of predesignated vertices called *portals*. A Structure Theorem guarantees the existence of a nearly optimal solution (one that has length at most $(1+\epsilon)\mathrm{OPT}$) in this set.

4.1 The construction

In this work we do not give the details of the constructions due to [Borradaile et al. 2009; Borradaile et al. 2009]. We only mention some of the definitions in order to facilitate the presentation and proof.

The following steps are performed separately for each demand set. When working on $\mathcal{D}_i$ to build H_i, the set of terminals $Q = Q_i$ will include all the vertices appearing in the demand pairs of $\mathcal{D}_i$.

Step 1. **Building a mortar graph**: The mortar graph[2] (of a bounded-genus graph G) with respect to a subset Q of vertices, called *terminals*, is a subgraph G_M of G with the following properties, among others:

(1) $\ell(G_M) \leq \gamma(\epsilon, g)\,\mathrm{span}(Q)$, where $\gamma(\epsilon, g) = 2(8g+2)(\epsilon^{-1}+1)^2$ and $\mathrm{span}(Q)$ denotes the minimum length of a Steiner tree spanning terminals Q.

(2) A face B of G_M with all edges and vertices of G embedded inside it is called a brick. Every brick is planar.

(3) The boundary of each brick B consists of four *sides* W, N, E, S in clockwise order. The total length of all W- and E-boundaries (called *supercolumns*) is at most $\epsilon^2 \cdot \mathrm{span}(Q)$.

(4) All terminals Q fall on N- or S-boundaries.

Borradaile et al. [2009] show how to construct the mortal graph of a bounded-genus graph in $O(n \log n)$ time.

Step 2. **Designating portals**: For some θ that polynomially depends on ϵ^{-1} and g, at most θ vertices on the boundary of each brick B are designated as portals such that the distance between any two consecutive portals is no more than $\ell(\partial B)/2\theta$. (This can be done using a greedy algorithm: start with an arbitrary vertex as a portal, and then moving clockwise around the brick, iteratively find the farthest vertex on the boundary whose distance to the previously selected portal is within the prescribed limit.) The goal is to focus on *portal-respecting* solutions, i.e., those in which the solution crosses brick boundaries only at portals. To reduce the patching cost (of connecting the original solution's crossing points to portals), we pick θ to be large compared to the number of crossings a solution may have. In particular, Theorem 10.7 of [Borradaile et al. 2009] bounds the number of crossings by α which is a polynomial in g and ϵ^{-1}.

LEMMA 4.1. *For any forest F in a brick B, there exists a forest F' such that*

[2]The reader can refer to Definition 3.1 of [Borradaile et al. 2009] for a complete definition of the mortar graph.

(1) $\ell(F') \leq (1+\epsilon)\ell(F)$,

(2) F' *crosses*[3] *the boundary of* B *at most* α *times, and*

(3) *any two vertices on* N*- or* S*-boundaries of* B *connected by* F *are also connected by* F'.

Step 3. **Adding Steiner trees**: All edges of G_M belong to H_i. For each brick B and any selection of its portals $\Pi' \subseteq \Pi$, we add to H_i the optimal Steiner tree (not forest) spanning Π' and only using the boundary or the inside of B, assuming that the W- and E-boundaries of B have length zero. This can be done in time polynomial in θ using the algorithm of [Erickson et al. 1987] since all these terminals lie on the infinite face of a planar graph.

Notice that for fixed ϵ and g, there are at most a constant number of portals, hence constant number of such Steiner trees, and the length of each is at most the length of the boundary of the brick.

4.2 The analysis

Next we prove two properties of the spanner: a bound on its length in Lemma 4.2 and its spanning property in Lemma 4.3. The following lemma is the main piece in proving the shortness property.

LEMMA 4.2. *The length of* H_i *is at most* $f(\epsilon, g)\ell(T_i)$ *for a universal certain function* $f(\epsilon, g)$.

PROOF. H_i is made up of the mortar graph G_M and the Steiner trees added in Step 3. We have $\ell(G_M) \leq \gamma(\epsilon, g)\,\mathrm{span}(Q_i) \leq \gamma(\epsilon, g)\ell(T_i)$. For a brick B in Step 3 we add at most 2^θ Steiner trees each of which has length no more than $\ell(\partial(B))$. Since an edge of G_M may appear in the boundary of two bricks, the total addition due to these trees is at most $2^{\theta+1}\ell(G_M)$. Therefore, $\ell(H_i) \leq (2^{\theta+1}+1)\ell(G_M) \leq (2^{\theta+1}+1)\gamma(\epsilon, g)\ell(T_i) = f(\epsilon, g)\ell(T_i)$. □

Finally we prove the spanning property of H. Recall that H is formed by the union of the graphs H_i constructed above.

LEMMA 4.3. $\mathrm{OPT}_{\mathcal{D}}(H) \leq (1+c'\epsilon)\mathrm{OPT}_{\mathcal{D}}(G_{in})$ *for a universal constant* $c' > 0$.

PROOF. Take the optimal solution OPT. Find forests OPT_i satisfying demands $\mathcal{D}_i$, i.e., $\ell(\mathrm{OPT}_i) = \mathrm{OPT}_{\mathcal{D}_i}(G_{in})$. We can apply Theorem 1.3 to get $\sum_i \ell(\mathrm{OPT}_i) \leq (1+\epsilon)\mathrm{OPT}$.

Consider one OPT_i that serves the respective set of demands $\mathcal{D}_i$. Add the set of all supercolumns of H_i to OPT_i to get OPT_i^1. Recall the total length of these supercolumns is at most $\epsilon^2 \cdot \mathrm{span}(Q_i) \leq \epsilon^2\ell(T_i)$. Next, use Lemma 4.1 to replace the intersection of OPT_i^1 and each brick with another forest having the properties of the lemma. Let OPT_i^2 be the new forest. The length of the solution increases to no more than a $1+\epsilon$ factor. Furthermore, as a result, OPT_i^2 crosses each brick at most α times. We claim, provided that θ is sufficiently large compared to α, we

[3]The actual proof bounds the number of "joining vertices" in a brick, however, the number of crossings is no more than the number of joining vertices, and the number of crossings is indeed what is required for making the solution portal-respecting.

can ensure that moving these intersection points to the portals introduces no more than an ϵ factor in the length.

Consider a brick B with boundaries W, N, E, S. Connect each intersection point of the brick to its closest portal. Each connection on a brick B moves by at most $\ell(\partial(B))/\theta$. The total movement of each brick is at most $\alpha\ell(\partial(B))/\theta$ which is no more than $\epsilon^2\ell(\partial(B))/\gamma(\epsilon, g)$ if $\theta \geq \alpha\epsilon^{-2}\gamma(\epsilon, g)$. Hence, the total additional length for all bricks of H_i is bounded by $2\epsilon^2\ell(T_i)$.

Finally, we replace the forests inside each brick B by the Steiner trees provisioned in the last step of our spanner construction. Take a brick B with the set of portals Π. Let $K_1, K_2, \ldots$ be the connected components of OPT_i^2 inside B. Each intersection point is connected to a portal of B. Replace each K_j by the optimal Steiner tree corresponding to this subset of portals. This procedure does not increase the length and produces a graph OPT_i^3.

Clearly, OPT_i^3 satisfies all the demands in $\mathcal{D}_i$. Thus, the union of all forests OPT_i^3, henceforth referred to as OPT^*, gives a solution for the given Steiner forest instance. It only remains to bound the length of OPT^*. We have

$$\ell(\mathrm{OPT}^*) \leq \sum_i \ell(\mathrm{OPT}_i^3), \tag{4}$$

which may be strict due to the presence of common edges between different OPT_i^3 forests. Replacing the Steiner trees by the optimal Steiner trees between portals cannot increase the length, so, the only length increase comes from connections to portals. Thus, we get

$$\ell(\mathrm{OPT}^*) \leq \sum_i \left[\ell(\mathrm{OPT}_i^2) + 2\epsilon^2\ell(T_i)\right]. \tag{5}$$

From the above discussion,

$$\ell(\mathrm{OPT}_i^2) \leq (1+\epsilon)\ell(\mathrm{OPT}_i^1), \qquad \text{by Lemma 4.1, and} \tag{6}$$

$$\ell(\mathrm{OPT}_i^1) \leq \ell(\mathrm{OPT}_i) + \epsilon^2\ell(T_i) \qquad \text{due to supercolumns' length.} \tag{7}$$

Hence, the length of OPT^* is

$$\begin{aligned}
\ell(\mathrm{OPT}^*) &\leq \sum_i \left[(1+\epsilon)\ell(\mathrm{OPT}_i) + \epsilon^2(3+\epsilon)\ell(T_i)\right] && \text{by (5), (6) and (7)}\\
&= \sum_i \left[(1+\epsilon)\ell(\mathrm{OPT}_i)\right] + \sum_i \left[\epsilon^2(3+\epsilon)\ell(T_i)\right]\\
&\leq (1+\epsilon)^2\mathrm{OPT} + \epsilon^2(3+\epsilon)\sum_i \ell(T_i) && \text{by Theorem 1.3}\\
&\leq (1+\epsilon)^2\mathrm{OPT} + \epsilon^2(3+\epsilon)(4/\epsilon+2)\mathrm{OPT} && \text{by Theorem 1.3}\\
&= \left[1 + 14\epsilon + 11\epsilon^2 + 2\epsilon^3\right]\mathrm{OPT}\\
&\leq (1+c'\epsilon)\mathrm{OPT},
\end{aligned}$$

if we pick $c' = 27$ and assume $\epsilon \leq 1$. □

Notice that the construction will produce a subset of the original graph, hence the bound on genus carries over.

4.3 Running time

The spanner can be constructed in time $O(n^2 \log n)$, where the dominant term comes from `PC-Clustering`. Borradaile et al. [2009] show how to construct the mortar graph in time $O(n \log n)$. The rest of the spanner construction can be done in $O(n \log n)$ time [Borradaile et al. 2009].

5. THE PTAS FOR BOUNDED-GENUS STEINER FOREST

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 et al. 2007].

Theorem 5.1 ([Demaine et al. 2007]). *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 $O(g^2 k)$. 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). According to [Borradaile, Demaine, and Tazari 2009], the running time of the above procedure can be improved to $O(n \log n)$ using the techniques in [Cabello and Chambers 2007] (assuming constant g).

We can now prove the main theorem of this work. Algorithm 3 (`SF-PTAS`) shows the steps of the PTAS.

Algorithm 3 `SF-PTAS`

Input: bounded-genus 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 $k \leftarrow 2f(\epsilon)/\bar{\epsilon}$.
3: Let $\epsilon \leftarrow \min(1, \bar{\epsilon}/6)$.
4: Using Theorem 5.1, partition the edges of $H$ into $E_1, \ldots, E_k$.
5: Let $i^* \leftarrow \arg\min_i \ell(E_i)$.
6: Find a $(1+\epsilon)$-approximate Steiner forest $F^*$ of $\mathcal{D}$ in $H/E_{i^*}$ via Theorem 1.5.
7: Output $F^* \cup E_{i^*}$.
```

Proof of Theorem 1.1. Given are bounded-genus graph G_{in}, and the set of demand pairs $\mathcal{D}$. We build a Steiner forest spanner H using Theorem 1.2. For a suitable value of k whose precise value will be fixed below, we apply Theorem 5.1 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 $O(g^2 k)$.

Theorem 1.5 allows us to find a solution OPT* corresponding to H^*. Adding the edge set E_{i^*} clearly produces a solution for H whose length is at most $(1+\epsilon)\mathrm{OPT}_{\mathcal{D}}(H) + \ell(H)/k$. Letting $\epsilon = \min(1, \bar{\epsilon}/6)$ and $k = 2f(\epsilon)/\bar{\epsilon}$ guarantees that the length of this solution is

$$\begin{aligned}
&\le (1+\epsilon)^2 \mathrm{OPT}_{\mathcal{D}}(G_{in}) + \ell(H)/k && \text{by Theorem 1.2}\\
&\le (1+\epsilon)^2 \mathrm{OPT}_{\mathcal{D}}(G_{in}) + \frac{\bar{\epsilon}}{2}\mathrm{OPT}_{\mathcal{D}}(G_{in}) && \text{by Theorem 1.2 and choice of } k\\
&(1+\bar{\epsilon})\mathrm{OPT}_{\mathcal{D}}(G_{in}) && \text{by the choice of } \epsilon. \quad \square
\end{aligned}$$

The running time of the algorithm excluding 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 bounded-treewidth instances is not bounded by a low-degree polynomial; rather, k and ϵ appear in the exponent of the polynomial. Were we able to improve the running time of this procedure, we would obtain a PTAS that runs in time $O(n^2 \log n)$.

6. PTAS FOR GRAPHS OF BOUNDED TREEWIDTH

The purpose of this section is to prove Theorem 1.5 by presenting a PTAS for *Steiner forest* on graphs of bounded treewidth.

6.1 Groups

We define a notion of group that will be crucial in the description of 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 6.1. *Let T be a Steiner tree of $X \subseteq V(G)$ with length W. For every $\epsilon > 0$, there is a set $S \subseteq X$ of $O(1 + 1/\epsilon)$ vertices such that $X = \mathcal{G}_G(X, S, \epsilon W)$.*

Proof. Let us select vertices s_1, s_2, ... from X as long as possible, with the requirement that the distance of s_i is more than ϵW from every s_j, $1 \le j < i$. Suppose that s_t is the last vertex selected this way. We claim that $t \le 1 + 2/\epsilon$. Consider a shortest closed tour in G that visits the vertices $s_1, \dots, s_t$ (not necessarily in the order of their indices). As the distance between any two such vertices is more than ϵW, the total length of the tour is more than $t\epsilon W$ (assuming that $t > 1$). On the other hand, all these vertices are on the tree T and it is well known that there is a closed tour that visits every vertex of the tree in such a way that every edge of the tree is traversed exactly twice and no other edge of the graph is used. Hence the shortest tour has length at most $2W$ and $t \le 2/\epsilon$ follows. $\square$

The following consequence of the definition of group is easy to see.

Proposition 6.2. *If S_1, S_2, X_1, X_2 are subsets of vertices of G and r_1, r_2 are real numbers, then*

$$\mathcal{G}_G(X_1, S_1, r_1) \cup \mathcal{G}_G(X_2, S_2, r_2) \subseteq \mathcal{G}_G(X_1 \cup X_2, S_1 \cup S_2, \max\{r_1, r_2\}).$$

6.2 Conforming solutions

Let $(T, \mathcal{B})$ be a rooted nice tree decomposition of width k, let I be the nodes of T, and let $\mathcal{B} = \{B_i \mid i \in I\}$ be the bags of the decomposition. For every $i \in I$, let V_i be the set of vertices appearing in B_i or in the bag of 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

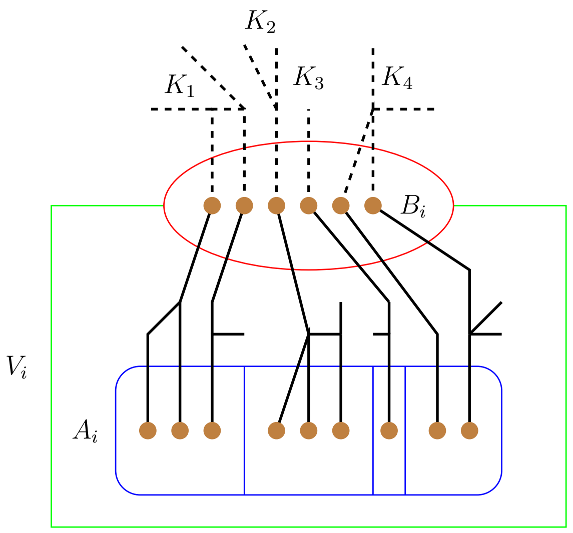

Fig. 2. The 4 components K_1, K_2, K_3, K_3 of F partition A_i into 4 classes. Note that the restriction of F to V_i has 6 components.

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 $i \in I$: 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 than the partition defined by the components of the restriction of F to G_i. See Figure 4 for an example.

Let $\Pi = (\Pi_i)_{i \in I}$ be a collection such that Π_i is a set of partitions of A_i. If some Steiner forest F satisfies $\pi_i(F) \in \Pi_i$ for every $i \in I$, then we say that F *conforms* to Π. The aim of this subsection is to give an algorithm for bounded treewidth graphs that finds a minimum-length solution conforming to a given Π. For fixed k, the running time is polynomial in the size of the graph and the size of the collection Π on a graph with treewidth at most k. In Section 6.3, we construct a polynomial-size collection Π such that there is a $(1+\epsilon)$-approximate solution that conforms to Π. Putting together these two results, we get a PTAS for the Steiner forest problem on bounded treewidth graphs.

Lemma 6.3. *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-length Steiner forest conforming to Π.*

The proof of Lemma 6.3 follows the standard dynamic programming approach, but it is not completely trivial. First, we use a technical trick that makes the presentation of the dynamic programming algorithm simpler. We can assume that every terminal vertex v has degree 1: otherwise, moving the terminal to a new degree 1 vertex v' attached to v with an edge vv' having length 0 does not change the problem and does not increase treewidth. Thus by Lemma 2.1, it can be assumed that we have a nice tree decomposition $(T, \mathcal{B})$ of width at most k where

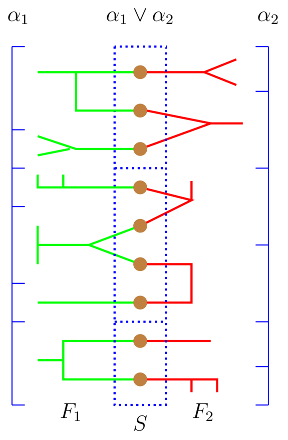

Fig. 3. Forest F_1 induces partition α_1 on S, forest F_2 induces partition α_2, the union of the two forests induces the partition $\alpha_1 \vee \alpha_2$.

no terminal vertex is introduced and the join nodes contain no terminal vertices; this assumption simplifies the presentation. For the rest of the section, we fix such a tree decomposition and notation V_i, A_i, etc. refer to this fixed decomposition.

Let us introduce terminology and notation concerning partitions. A partition α of a set S can be considered as an equivalence relation on S. Hence we use notation $(x, y) \in \alpha$ to say that x and y are in the same class of α. We denote by x^α the class of α that contains element x.

If F is a subgraph of G and $S \subseteq V(G)$, then F *induces* a partition α of S: $(x, y) \in \alpha$ if and only if x and y are in the same component of F (and every $x \in S \setminus V(F)$ forms its own class). We say that partition α is *finer* than partition β if $(x, y) \in \alpha$ implies $(x, y) \in \beta$; in this case, β is *coarser* than α. If $\alpha = \beta$, then α is both finer and coarser than β. We denote by $\alpha_1 \vee \alpha_2$ the unique finest partition α coarser than both α_1 and α_2. This definition is very useful in the following situation. Let F_1, F_2 be subgraphs of G, and suppose that F_1 and F_2 induce partitions α_1 and α_2 of a set $S \subseteq V(G)$, respectively. If F_1 and F_2 intersect only in S, then the partition induced by the union of F_1 and F_2 is exactly $\alpha_1 \vee \alpha_2$ (see Figure 3). Let β_i be a partition of B_i for some $i \in I$ and let F_i be a subgraph of $G[V_i]$. Then we denote by $F_i + \beta_i$ the graph obtained from F_i by adding a new edge xy for every $(x, y) \in \beta_i$. Note that $F_i + \beta_i$ is not necessarily a subgraph of $G[V_i]$.

Following the usual method of designing algorithms for bounded-treewidth graphs, we define several subproblems for each node $i \in I$. A subproblem at node i corresponds to finding a subgraph F_i in G_i satisfying certain properties: informally speaking, F_i is supposed to be the restriction of a Steiner forest F to V_i. The properties defining a subproblem prescribe how F_i should look from the "outside world" (i.e., from the part of G outside V_i) and they contain all the information necessary for deciding whether F_i can be extended, by edges outside V_i, to a conforming solution. Let us discuss briefly and informally what information these prescriptions should contain. Clearly, the edges of F_i in B_i and the way F_i connects the vertices of B_i (i.e., the partition α of B_i induced by F_i) is part of this information. Furthermore, the way F_i partitions A_i should also be part of this information. However, there is a subtle detail that makes the description of our algorithm significantly more technical. The definition of $\pi_i(F) = \pi$ means that the components of F partition A_i in a certain way. But the restriction F_i of F to V_i might induce a finer partition of A_i than π: it is possible that two components of F_i are in the same component of F (see Figure 4). This means that we cannot require that the partition of A_i induced by F_i belongs to Π. We avoid this problem by "imagining" the partition β of B_i induced by the full solution F, and require that F_i partition A_i according to π if each class of β becomes connected somehow. In other words, instead of requiring that F_i itself partitions A_i in a certain way, we require that $F_i + \beta$ induce a certain partition.

Formally, each subproblem P is defined by a tuple $(i, H, \pi, \alpha, \beta, \mu)$, where

(S1) $i \in I$ is a node of T,
(S2) H is a spanning subgraph of $G[B_i]$ (i.e., contains all vertices of $G[B_i]$),
(S3) $\pi \in \Pi_i$ is a partition of A_i,
(S4) α, β are partitions of B_i, β is coarser than α, and α is coarser than the partition induced by the components of H, and
(S5) μ is an injective mapping from the classes of π to the classes of β.

The solution $c(i, H, \pi, \alpha, \beta, \mu)$ of a subproblem P is the minimum length of a subgraph F_i of $G[V_i]$ satisfying all of the following requirements:

(C1) $F_i[B_i] = H$ (which implies $B_i \subseteq V(F_i)$).
(C2) α is the partition of B_i induced by F_i.
(C3) The partition of A_i induced by $F_i + \beta$ is π.
(C4) For every descendant d of i (including $d = i$), the partition of A_d induced by $F_i + \beta$ belongs to Π_d.
(C5) If there is a terminal pair (x_1, x_2) with $x_1, x_2 \in V_i$, then they are connected in $F_i + \beta$.
(C6) Every $x \in A_i$ is in the component of $F_i + \beta$ containing $\mu(x^\pi)$.

We solve these subproblems by bottom-up dynamic programming. Let us discuss how to solve a subproblem depending on the type of i.

Leaf nodes i. If i is a leaf node, then the value of the solution is trivially 0.

Join node i having children i_1, i_2. Note that A_{i_1} and A_{i_2} are disjoint: the vertices of a join node are not terminal vertices. The set A_i is a subset of $A_{i_1} \cup A_{i_2}$ and it may be a proper subset: if there is a pair (x, y) with $x \in A_{i_1}$, $y \in A_{i_2}$, then x or y might not be in A_i.

We show that the value of the subproblem is

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(J1),(J2),(J3),(J4)} (c(i_1, H, \pi^1, \alpha^1, \beta, \mu^1) + c(i_2, H, \pi^2, \alpha^2, \beta, \mu^2) - \ell(H)), \quad (8)$$

where the minimum is taken over all tuples satisfying, for $p = 1, 2$, all of the following conditions:

(J1) $\alpha^1 \vee \alpha^2 = \alpha$.

(J2) π and π^p are the same on $A_{i_p} \cap A_i$.

(J3) For every $v \in A_{i_p} \cap A_i$, $\mu(v^\pi) = \mu^p(v^{\pi^p})$ (note that the classes of β are the domain of both μ and μ^p).

(J4) If there is a terminal pair (x_1, x_2) with $x_1 \in A_1$ and $x_2 \in A_2$, then $\mu^1(x_1^{\pi^1}) = \mu^2(x_2^{\pi^2})$.

We will use the following observation repeatedly. Let F be a subgraph of G_i and let $F^p = F[V_{i_p}]$. Suppose that F induces partition α on B_i and β is a coarser partition than α. Then two vertices of V_{i_p} are connected in $F + \beta$ if and only if they are connected in $F^p + \beta$. Indeed, F^{3-p} does not provide any additional connectivity compared to $F^p + \beta$: as β is coarser than α_{3-p}, if two vertices of B_i are connected by a path in F^{3-p}, then they are already adjacent in $F^p + \beta$.

Proof of (8) *left* $\leq$ (8) *right:*

Let $P_1 = (i_1, H, \pi^1, \alpha^1, \beta, \mu^1)$ and $P_2 = (i_2, H, \pi^2, \alpha^2, \beta, \mu^2)$ be subproblems minimizing the right-hand side of (8), and let F^1 and F^2 be optimal solutions of P_1 and P_2, respectively. Let F be the union of subgraphs F^1 and F^2. It is clear that the length of F is exactly the right-hand side of (8): the common edges of F^1 and F^2 are exactly the edges of H. We show that F is a solution of P, i.e., F satisfies requirements (C1)–(C6).

(C1): Follows from $F^1[B_i] = F^2[B_i] = F[B_i] = H$.

(C2): Follows from (J1) and from the fact that F^1 and F^2 intersect only in B_i.

(C3): First consider two vertices $x, y \in A_{i_p} \cap A_i$. Vertices x and y are connected in $F + \beta$ if and only if they are connected in $F^p + \beta$. By (C3) for F^p, this is equivalent to $(x, y) \in \pi^p$, which is further equivalent to $(x, y) \in \pi$ by (J2). Now suppose that $x \in A_{i_1} \cap A_i$ and $y \in A_{i_2} \cap A_i$. In this case, x and y are connected in $F + \beta$ if and only if there is a vertex of B_i reachable from x in $F^1 + \beta$ and from y in $F^2 + \beta$, or in other words, $\mu^1(x^{\pi^1}) = \mu^2(y^{\pi^2})$. By (J3), this is equivalent to $\mu(x^\pi) = \mu(y^\pi)$, or $(x, y) \in \pi$ (as μ is injective).

(C4): If d is a descendant of i_p, then the statement follows using that (C4) holds for solution F^p of P^p and the fact that for every descendant d of i_p, $F_i + \beta$ and $F + \beta$ induce the same partition of A_d. For $d = i$, the statement follows from the previous paragraph, i.e., from the fact that $F + \beta$ induces partition $\pi \in \Pi_i$ on A_i.

(C5): Consider a pair (x_1, x_2). If $x_1, x_2 \in V_{i_1}$ or $x_1, x_2 \in V_{i_2}$, then the statement follows from (C5) on F^1 or F^2. Suppose now that $x_1 \in V_{i_1}$ and $x_2 \in V_{i_2}$; in this case, we have $x_1 \in A_{i_1}$ and $x_2 \in A_{i_2}$. By (C6) on F^1 and F^2, x_p is connected to $\mu^p(x_p^{\pi^p})$ in $F^p + \beta$. By (J4), we have $\mu^1(x_1^{\pi^1}) = \mu^2(x_2^{\pi^2})$, hence x_1 and x_2 are connected to the same class of β in $F + \beta$.

(C6): Consider an $x \in A_i$ that is in A_{i_p}. By condition (C6) on F^p, we have that x is connected in $F^p + \beta$ (and hence in $F + \beta$) to $\mu^p(v^{\pi^p})$, which equals $\mu(v^\pi)$ by (J3).

Proof of (8) *left* $\geq$ (8) *right:*

Let F be a solution of subproblem $(i, H, \pi, \alpha, \beta, \mu)$ and let F^p be the subgraph of F induced by V_{i_p}. To prove the inequality, we need to show three things. First, we have to define two tuples $(i_1, H, \pi^1, \alpha^1, \beta, \mu^1)$ and $(i_2, H, \pi^2, \alpha^2, \beta, \mu^2)$ that are subproblems, i.e., they satisfy (S1)–(S5). Second, we show that (J1)–(J4) hold for these subproblems. Third, we show that F^1 and F^2 are solutions for these subproblems (i.e., (C1)–(C6)), hence they can be used to give an upper bound on the right-hand side that matches the length of F.

Let α^p be the partition of B_i induced by the components of F^p; as F^1 and F^2 intersect only in B_i, we have $\alpha = \alpha^1 \vee \alpha^2$, ensuring (J1). Since β is coarser than α, it is coarser than both α^1 and α^2. Let π^p be the partition of A_{i_p} defined by $F + \beta$; we have $\pi_p \in \Pi_{i_p}$ by (C4) for F. Furthermore, by (C3) for F, π is the partition of A_i induced by $F + \beta$, hence it is clear that π and π^p are the same on $A_{i_p} \cap A_i$, so (J2) holds. This also means that $F + \beta$ (or equivalently, $F^p + \beta$) connects a class of π^p to exactly one class of β; let μ^p be the corresponding mapping from the classes of π^p to β. Now (J4) is immediate. It is clear that the tuple $(i_p, H, \pi^p, \alpha^p, \beta, \mu^p)$ satisfies (S1)–(S5).

We show that F^p is a solution of subproblem $(i_p, H, \pi^p, \alpha^p, \beta, \mu^p)$. As the edges of H are shared by F^1 and F^2, it will follow that the right-hand side of (8) is not greater than the left-hand side.

(C1): Obvious from the definition of F^1 and F^2.

(C2): Follows from the way α^p is defined.

(C3): Follows from the definition of π^p, and from the fact that $F + \beta$ and $F^p + \beta$ induce the same partition on A_{i_p}.

(C4): Follows from (C4) on F and from the fact that $F + \beta$ and $F^p + \beta$ induces the same partition on A_d.

(C5): Suppose that $x_1, x_2 \in V_{i_p}$. Then by (C5) for F, x_1 and x_2 are connected in $F + \beta$, hence they are connected in $F_i + \beta$ as well.

(C6): Follows from the definition of μ^p.

Introduce node i of vertex v. Let j be the child of i. Since v is not a terminal vertex, we have $A_j = A_i$. Let F' be a subgraph of $G[V_j]$ and let F_S be obtained from F' by adding vertex v to F' and making v adjacent to $S \subseteq B_j$. If α' is the partition of B_j induced by the components of F', then we define the partition $\alpha'[v, S]$ of B_i to be the partition obtained by joining all the classes of α' that intersect S and adding v to this new class (if $S = \emptyset$, then $\{v\}$ is a class of $\alpha'[v, S]$). It is clear that $\alpha'[S, v]$ is the partition of B_i induced by F_S.

We show that the value of a subproblem is given by

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(I1),(I2),(I3)} c(j, H[B_j], \pi, \alpha', \beta', \mu') + \sum_{xv \in E(H)} \ell(xv), \qquad (9)$$

where the minimum is taken over all tuples satisfying all of the following:

(I1) $\alpha = \alpha'[v, S]$, where S is the set of neighbors of v in H.

(I2) β' is β restricted to B_j.

(I3) For every $x \in A_i$, $\mu(x^\pi)$ is the class of β containing $\mu'(x^\pi)$.

Proof of (9) *left* $\leq$ (9) *right:*

Let F' be an optimal solution of subproblem $P' = (j, H[B_j], \pi, \alpha', \beta', \mu')$. Let F be the graph obtained from F' by adding to it the edges of H incident to v; it is clear that the length of F is exactly the right-hand side of (9). Let us verify that (C1)–(C6) hold for F.

(C1): Immediate.

(C2): Holds because of (I1) and the way $\alpha'[v, S]$ was defined.

(C3)–(C5): Observe that $F + \beta$ connects two vertices of V_j if and only if $F' + \beta'$ does. Indeed, if a path in $F + \beta$ connects two vertices via vertex v, then the two neighbors x, y of v on the path are in the same class of β as v (using that α and β are coarser than the partition induced by H), hence (I2) implies that x, y are in the same class of β' as well. In particular, for every descendant d of i, the components of $F + \beta$ and the components of $F' + \beta$ give the same partition of A_d.

(C6): Follows from (C6) for F' and from (I3).

Proof of (9) *left* $\geq$ (9) *right:*

Let F be a solution of subproblem $(i, H, \pi, \alpha, \beta, \mu)$ and let F' be the subgraph of F induced by V_j. We define a tuple $(j, H[B_j], \pi, \alpha', \beta', \mu')$ that is a subproblem, show that it satisfies (I1)–(I3), and that F' is a solution of this subproblem.

Let α' be the partition of V_j induced by F' and let β' be the restriction of β on B_j; these definitions ensure that (I1) and (I2) hold. Let $\mu'(x^\pi) = \mu(x^\pi) \setminus \{v\}$, which is a class of β'; clearly, this ensures (I3). Note that this is well defined, as it is not possible that $\mu(x^\pi)$ is a class of β consisting of only v: by (C6) for F, this would mean that v is the only vertex of B_i reachable from x in F. Since v is not a terminal vertex, $v \neq x$, thus if v is reachable from x, then at least one neighbor of v has to be reachable from x as well.

Let us verify that (S1)–(S5) hold for the tuple $(j, H[B_j], \pi, \alpha', \beta', \mu')$. (S1) and (S2) clearly hold. (S3) follows from the fact that (C4) holds for F and $A_i = A_j$. To see that (S4) holds, observe that $(x, y) \in \alpha'$ implies $(x, y) \in \alpha$, which implies $(x, y) \in \beta$, which implies $(x, y) \in \beta'$. (S5) is clear from the definition of μ'.

The difference between the length of F and the length of F' is exactly $\sum_{xv \in E(H)} \ell(xv)$. Thus to show that the left-hand side of (9) is at most the right-hand side of (9), it is sufficient to show that F' is a solution of subproblem $(j, H[B_j], \pi, \alpha', \beta', \mu')$.

(C1)–(C2): Obvious.

(C3)–(C5): As in the other direction, follow from the fact that $F' + \beta'$ induces the same partition of V_j as $F + \beta$.

(C6): By the definition of μ', it is clear that $\mu'(x^\pi)$ is exactly the subset of B_j that is reachable from x in $F + \beta$ and hence in $F' + \beta'$.

Forget node i of vertex v. Let j be the child of i. We have $V_i = V_j$ and hence $A_i = A_j$. We show that the value of a subproblem is given by

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(F1),(F2),(F3),(F4)} c(j, H', \pi, \alpha', \beta', \mu'), \tag{10}$$

where the minimum is taken over all tuples satisfying all of the following:

(F1) $H'[B_i] = H$.

(F2) α is the restriction of α' to B_i.

(F3) β is the restriction of β' to B_i and $(x, v) \in \beta'$ if and only if $(x, v) \in \alpha'$.

(F4) For every $x \in A_i$, $\mu(x^\pi)$ is the (nonempty) set $\mu'(x^\pi) \setminus \{v\}$ (which implies that $\mu'(x^\pi)$ contains at least one vertex of B_i).

Proof of (10) *left* $\leq$ (10) *right:*

Let F be a solution of $(j, H', \pi, \alpha', \beta', \mu')$. We show that F is a solution of $(j, H, \pi, \alpha, \beta, \mu)$ as well.

(C1): Clear because of (F1).

(C2): Clear because of (F2).

(C3)–(C5): We only need to observe that $F + \beta$ and $F + \beta'$ have the same components: since by (F3), $(x, v) \in \beta'$ implies $(x, v) \in \alpha'$, the neighbors of v in $F + \beta'$ are reachable from v in F, thus $F + \beta'$ does not add any further connectivity compared to $F + \beta$.

(C6): Observe that if $\mu'(x^\pi)$ are the vertices of B_j reachable from x in $F + \beta'$, then $\mu(x^\pi) = \mu'(x^\pi) \setminus \{v\}$ are the vertices of B_i reachable from x in $F + \beta'$. We have already seen that $F + \beta$ and $F + \beta'$ have the same components, thus the nonempty set $\mu(x^\pi)$ is indeed the subset of B_i reachable from x in $F + \beta$. Furthermore, by (F3), β is the restriction of β' on B_i, thus if $\mu'(x^\pi)$ is a class of β', then $\mu(x^\pi)$ is a class of β.

Proof of (10) *left* $\geq$ (10) *right:*

Let F be a solution of $(j, H, \pi, \alpha, \beta, \mu)$. We define a tuple $(j, H', \pi, \alpha', \beta', \mu')$ that is a subproblem, we show that (F1)–(F3) hold, and that F is a solution of this subproblem.

Let us define $H' = F[B_j]$ and let α' be the partition of B_j induced by the components of F; these definitions ensure that (F1) and (F2) hold. We define β' as the partition obtained by extending β to B_j such that v belongs to the class of β that contains a vertex $x \in B_i$ with $(x, v) \in \alpha'$ (as β is coarser than the partition induced by H, there is at most one such class; if there is no such class, then we let $\{v\}$ be a class of β'). It is clear that (F3) holds for this β'. Let us note that $F + \beta$ and $F + \beta'$ have the same connected components: if $(x, v) \in \beta'$, then x and v are connected in F. Let $\mu'(x^\pi)$ be the subset of B_j reachable from x in $F + \beta'$ (or equivalently, in $F + \beta$). It is clear that $\mu(x^\pi) = \mu'(x^\pi) \setminus \{v'\}$ holds, hence (F4) is satisfied.

Let us verify first that (S1)–(S5) hold for $(j, H', \pi, \alpha', \beta', \mu')$. (S1) and (S2) clearly holds. (S3) follows from the fact that (S3) holds for $(i, H, \pi, \alpha, \beta, \mu)$ and $A_i = A_j$. To see that (S4) holds, observe that if $x, y \in B_i$, then $(x, y) \in \alpha'$ implies $(x, y) \in \alpha$, which implies $(x, y) \in \beta$, which implies $(x, y) \in \beta'$. Furthermore, if $(x, v) \in \alpha'$, then $(x, v) \in \beta'$ by the definition of β. (S5) is clear from the definition of μ'.

We show that F is a solution of $(j, H', \pi, \alpha', \beta', \mu')$.

(C1): Clear from the definition of H'.

(C2): Clear from the definition of α'.

(C3)–(C5): Follow from the fact that $F + \beta$ and $F + \beta'$ have the same connected components.

(C6): Follows from the definition of μ'.

6.3 Constructing the partitions

Recall that the collection $\Pi = (\Pi_i)_{i\in I}$ contains a set of partitions Π_i for each $i \in I$. We construct these sets Π_i in the following way. Each partition in Π_i is defined by a sequence $((S_1, r_1), \dots, (S_p, r_p))$ of at most $k+1$ pairs and a partition ρ of $\{1, \dots, p\}$. The pair (S_j, r_j) consists of a set S_j of $O((k+1)(1+1/\epsilon))$ vertices of G_i and a nonnegative real number r_j, which equals the distance between two vertices of G. This means that there are at most $|V(G)|^{O((k+1)(1+1/\epsilon))} \cdot |V(G)|^2$ possible pairs (S_j, r_j) and hence at most $|V(G)|^{O((k+1)^2(1+1/\epsilon))}$ 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 near-optimal 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 a total ordering of the vertices in a bag. We can extend this relation to connected subsets 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 $u \in K_1$, or 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 6.4. *There is a $(1 + k\epsilon)$-approximate solution conforming to* Π.

Proof. Let F be a minimum-length 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 length at most $(1+k\epsilon)\ell(F)$. We need a delicate charging argument to show that the total increase of the length is at most $k\epsilon \cdot \ell(F)$ during the procedure. In each step, we charge the increase of the length 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

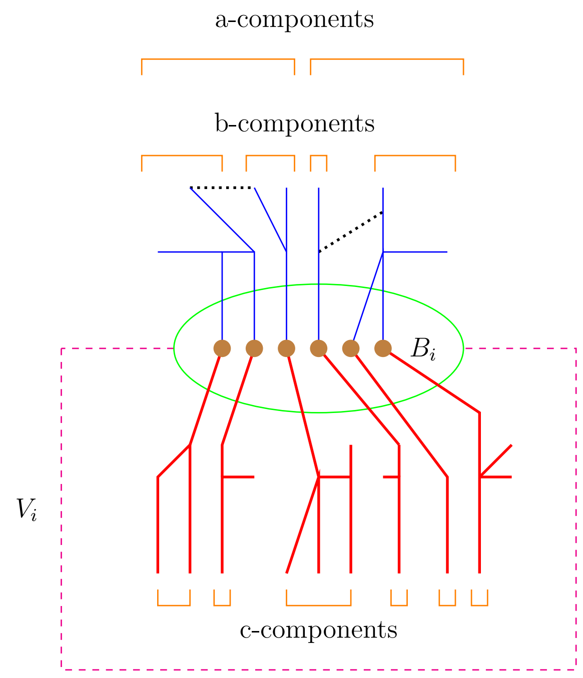

Fig. 4. The original solution F (plain and strong edges) consists of 4 b-components. Restricting F to V_i yields 6 c-components (strong edges). Forest F', which is obtained from F by adding the two dotted edges, has two a-components.

$K_1 < K_2$ 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 length increase of at most $\epsilon \cdot \ell(K_1)$ on the pair (K_1, K_2), thus the total increase is at most $k\epsilon \cdot \ell(F)$. 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 length at most $\ell(K_j)$. Thus by Lemma 6.1 and Proposition 6.2, there is a set $S_j \subseteq V(K_j)$ of at most $O((k+1)(1+1/\epsilon))$ vertices such that $A_{i,j} = \mathcal{G}_{G_i}(A_{i,j}, S_j, r_j)$ for some $r_j \le \epsilon \cdot \ell(K_j)$. 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 $d_{G_i}(u, v) \le r_{j^*} \le \epsilon \cdot \ell(K_{j^*})$. 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 length of F' by at most $\epsilon \cdot \ell(K_{j^*})$, which we charge on the pair (K_{j^*}, K_j) of *b*-components. 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. $\square$

7. ALGORITHM FOR SERIES-PARALLEL GRAPHS

A series-parallel graph is a graph that can be built using series and parallel composition. 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:

(1) An edge xy is a series-parallel graph.

(2) 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).*

(3) 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 1.4 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 [Bodlaender 1998]. 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 a series parallel graph. For $i = 1, \dots, m$, 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 as follows.

(1) Let a_i be the minimum length 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.
(2) Let G'_i be the graph obtained from G_i by identifying vertices x_i and y_i. Let b_i be the minimum length of a solution F of the instance $(G'_i, \mathcal{D}_i)$ with the additional requirement that every vertex of A_i is in the same component as $x_i = y_i$.
(3) For every $S \subseteq A_i$, let $f_i(S)$ be the minimum length 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 functions 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 7.1. 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 7.1, then the function is submodular. In the proof of Theorem 1.4, 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 7.2. *If a function $f : 2^A \to \mathbb{R}^+$ can be represented by (D, s, t, A) (in the sense of Definition 7.1), then f is submodular.*

PROOF OF THEOREM 1.4. 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 the length of each of the $k-1$ edges on the path formed by v and the new vertices is 0. Replacing vertex v in each pair involving it by one of the new vertices does not change the problem, and achieves the said property.

For every $i = 1, \ldots, m$, we compute the values a_i, and b_i, as well as a representation D_i of f_i. In the optimal solution F for the instance $(G_m, \mathcal{D})$, vertices x_m and y_m are either connected or not. Thus the length of the optimal solution 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-length s_m-t_m 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 $\{x_i, y_i\} \in \mathcal{D}_i$).

If G_i is not a single edge, then it is constructed from some G_{j_1} and G_{j_2} by either 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 first 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 of vertices 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 $\{u, v\} \in \mathcal{D}_i$ by the definition of $\mathcal{D}_i$, contradicting the assumption that F is a realization of $f_i(S)$. Therefore, for the set $X := X_1 \cup X_2$, we have

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

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 $\delta_{D_i}(X) \geq f_i(S)$. If $\delta_{D_i}(X) = \infty$, then this is trivially true, thus we assume that $\delta_{D_i}(X)$ is finite. For $p = 1, 2$, let $X_p = X \cap V(D_{j_p})$ and $S^p = A_{j_p} \cap X_p$. As $\delta_{D_i}(X)$ is finite, the infinite edges added in the construction of D_i do not appear on the boundary of X, hence $\delta_{D_i}(X) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2)$. Since D_{j_p} represents f_{j_p}, we know 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 $\ell(F) \geq f_i(S)$ holds by verifying that F satisfies all the requirements in the definition of $f_i(S)$. 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 $\{u, v\} \in \mathcal{D}_i \setminus \{\mathcal{D}_{j_1} \cup \mathcal{D}_{j_2}\}$ for some $u \in V(G_{j_1})$ and $v \in V(G_{j_2})$. Clearly, this means that $u \in A_{j_1}$ and $v \in A_{j_2}$. Suppose that F does not connect u and v, and, without loss of generality, assume that $u \in S^1$ and $v \notin S^2$. By definition of S^1 and S^2, it follows that $u \in X_1$ and $v \notin X_2$. This means that there is an edge $\overrightarrow{uv}$ of length ∞ in D_i, yielding $\delta_{D_i}(X) = \infty$, which contradicts our earlier assumption. Thus we can indeed assume $\ell(F) \geq f_i(S)$ and it follows that

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

This completes the argument for the parallel connection.

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 $a_i = a_{j_1} + a_{j_2}$ (i.e., 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 reversing the orientation of the edges and 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 observe that to compute b_i we can consider two cases: either μ is connected to x_{j_1} and y_{j_2} or not. We take the minimum of the two values. The first case is simply $\min a_{j_1} + b^R_{j_2}, b_{j_1} + a^R_{j_2} = \min a_{j_1} + b_{j_2}, b_{j_1} + a_{j_2}$, and the second case is $f'_i(A_i)$: 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.

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 $v \in T_1$ 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 5):

(1) $\overrightarrow{s_{j_1} \gamma_1}$ with length a_{j_2},

(2) $\overrightarrow{\gamma_1 \gamma_2}$ with length $f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2)$,

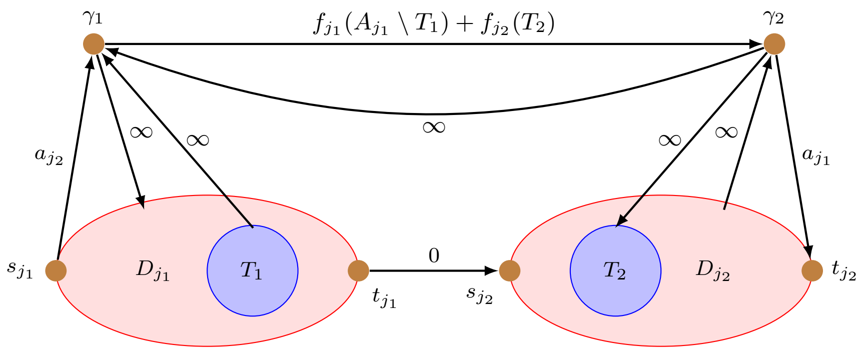

Fig. 5. Construction of D_i in a series connection.

(3) $\overrightarrow{\gamma_2 t_{j_2}}$ with length a_{j_1},
(4) $\overrightarrow{\gamma_2\gamma_1}$ with length ∞,
(5) $\overrightarrow{\gamma_1 v}$ with length ∞ for every $v \in V(D_{j_1})$,
(6) $\overrightarrow{v\gamma_2}$ with length ∞ for every $v \in V(D_{j_2})$,
(7) $\overrightarrow{v\gamma_1}$ with length ∞ for every $v \in T_1$, and
(8) $\overrightarrow{\gamma_2 v}$ 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 first show that D_i has an appropriate cut with value at most $f_i(S)$. 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} (recall that $x_i = x_{j_1}$, $\mu = y_{j_1} = x_{j_2}$, and $y_i = y_{j_2}$.

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 $X = V(D_{j_1}) \cup \{\gamma_1\}$: 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 $x_{j_2} = \mu$ and every vertex $A_i \setminus S = A_{j_2} \setminus (S \cup T_2)$ to y_{j_2}. This implies $\ell(F_2) \geq f_{j_2}((S \cap A_{j_2}) \cup T_2)$. 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 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 on 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

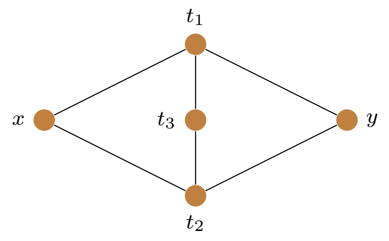

Fig. 6. The graph used in the proof Theorem 8.2.

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 subgraph 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 $\delta_{D_i}(X) = a_{j_1} + \delta_{D_{j_2}}(X_2)$ (where the term a_{j_1} comes from the edge $\overrightarrow{\gamma_2 t_{j_2}}$). 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 $F := F_1 \cup F_2$, and 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 $f_i(S) \leq \ell(F) = a_{j_1} + f_{j_2}(T_2 \cup (S \cap A_{j_2})) \leq \delta_{D_i}(X)$, as desired.

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

8. HARDNESS FOR TREEWIDTH 3

In this section, we show that *Steiner forest* is NP-hard on graphs with treewidth at most 3. Very recently, this has been proved independently by Gassner [2009], but our compact proof perhaps better explains what the reason is for the sharp difference between the series-parallel and the treewidth 3 cases.

Consider the graph in Figure 6 and let us define the function f analogously to the function f_i in Section 7: for every $S \subseteq \{1,2,3\}$, let $f(S)$ be the minimum length of a subgraph F where x and y are not connected, t_i is connected to x for every $i \in S$, and t_i is connected to y for every $i \in \{1,2,3\} \setminus S$; if there is no such subgraph F, then let $f(S) = \infty$. It is easy to see that $f(\{1,2\}) = f(\{2,3\}) = f(\{1,2,3\})$, while $f(\{2\}) = \infty$. Thus, unlike in the case of series-parallel graphs, this function is not submodular.

We use the properties of the function f defined in the previous paragraph to obtain a hardness proof in a more or less "automatic" way. Let us define the Boolean relation $R(a,b,c) := (a = c) \vee (b = c)$. Observe that for any $S \subseteq \{1,2,3\}$, we have $f(S) = 3$ if $R(1 \in S, 2 \in S, 3 \in S) = 1$ and $f(S) = \infty$ otherwise (here $1 \in S$ means the Boolean variable that is 1 if and only if $1 \in S$). Thus the gadget in Figure 6 in some sense represents the relation R and, as we shall see, this is sufficient to construct an NP-hardness proof.

An *R-formula* is a conjunction of clauses, where each clause is the relation R applied to some Boolean variables or to the constants 0 and 1, e.g., $R(x_1, 0, x_4) \wedge$

$R(0, x_2, x_1) \wedge R(x_3, x_2, 1)$. In the *R-SAT* problem, the input is an R-formula and it has to be decided whether the formula has a satisfying assignment.

LEMMA 8.1. *R-SAT is* NP*-complete.*

PROOF. Readers familiar with Schaefer's Dichotomy Theorem (more precisely, the version allowing constants [Schaefer 1978, Lemma 4.1]) can easily see that R-SAT is NP-complete. It is easy to verify that the relation R is neither weakly positive, weakly negative, affine, nor bijunctive. Thus the result of Schaefer immediately implies that *R-SAT* is NP-complete.

For completeness, we give a simple self-contained proof here. The reduction is from *Not-All-Equal 3SAT*[4], which is known to be NP-complete even if there are no negated literals [Schaefer 1978]. Given a *NAE-3SAT* formula, we replace each clause as follows. For each clause NAE(a, b, c), we introduce a new variable d and create the clauses $R(a, b, d) \wedge R(c, d, 0) \wedge R(c, d, 1)$. If $a = b = c$, then it is not possible that all three clauses are simultaneously satisfied (observe that the second and third clauses force $c \neq d$). On the other hand, if a, b, c do not have the same value, then all three clauses can be satisfied by an appropriate choice of d. Thus the transformation from *NAE-3SAT* to *R-SAT* preserves satisfiability. □

The main idea of the following proof is that we can simulate arbitrarily many R-relations by joining in parallel copies of the graph shown in Figure 6.

THEOREM 8.2. *Steiner forest is* NP*-hard on planar graphs with treewidth at most 3.*

PROOF. The proof is by reduction from *R-SAT*. Let ϕ be an R-formula having n variables and m clauses. We start the construction of the graph G by introducing two vertices v_0 and v_1. For each variable x_i of ϕ, we introduce a vertex x_i and connect it to both v_0 and v_1. We introduce 3 new vertices a_i, b_i, c_i corresponding to the i-th clause. Vertices a_i and b_i are connected to both v_0 and v_1, while c_i is adjacent only to a_i and b_i. If the i-th clause is $R(x_{i_1}, x_{i_2}, x_{i_3})$, then we add the 3 pairs $\{x_{i_1}, a_i\}$, $\{x_{i_2}, b_i\}$, $\{x_{i_3}, c_i\}$ to $\mathcal{D}$. If the clause contains constants, then we use the vertices v_0 and v_1 instead of the vertices x_{i_1}, x_{i_2}, x_{i_3}, e.g., the clause $R(0, x_{i_2}, 1)$ yields the pairs $\{v_0, a_i\}$, $\{x_{i_2}, b_i\}$, $\{v_1, c_i\}$. The length of every edge is 1. This completes the description of the graph G and the set of pairs $\mathcal{D}$.

We claim that the constructed instance of *Steiner forest* has a solution with $n + 3m$ edges if and only if the R-formula ϕ is satisfiable. Suppose that ϕ has a satisfying assignment f. We construct F as follows. If $f(x_i) = 1$, then let us add edge $x_i v_1$ to F; if $f(x_i) = 0$, then let us add edge $x_i v_0$ to F. For each clause, we add 3 edges to F. Suppose that the i-th clause is $R(x_{i_1}, x_{i_2}, x_{i_3})$. We add one of $a_i v_0$ or $a_i v_1$ to F depending on the value of $f(x_{i_1})$ and we add one of $b_i v_0$ or $b_i v_1$ to F depending on the value of $f(x_{i_2})$. Since the clause is satisfied, either $f(x_{i_3}) = f(x_{i_1})$ or $f(x_{i_3}) = f(x_{i_2})$; we add $c_i a_i$ or $c_i b_i$ to F, respectively (if $f(x_{i_3})$ is equal to both, then the choice is arbitrary). We proceed in an analogous manner

[4]In *Not-All-Equal 3SAT*, or *NAE-3SAT* for short, we are given a 3SAT instance with the extra restriction that a clause is not satisfied if *all* the literals in a clause are true. Similarly to *3SAT*, the clause is not satisfied if all the literals are false, either. Thus, the literals in each clause have to take both true and false values.

for clauses containing constants. It is easy to verify that each pair is in the same connected component of F.

Suppose now that there is a solution F with length $n + 3m$. At least one edge is incident with each vertex x_i, since it cannot be isolated in F. Each vertex a_i, b_i, c_i has to be connected to either v_0 or v_1, hence at least 3 edges of F are incident with these 3 vertices. As F has $n+3m$ edges, it follows that exactly one edge is incident with each x_i, and hence exactly 3 edges are incident with the set $\{a_i, b_i, c_i\}$. It follows that v_0 and v_1 are not connected in F. Define an assignment of ϕ by setting $f(x_i) = 0$ if and only if vertex x_i is in the same component of F as v_0. To verify that a clause $R(x_{i_1}, x_{i_2}, x_{i_3})$ is satisfied, observe that c_i is in the same component of F as either a_i or b_i. If c_i is in the same component as, say, a_i, then this component also contains x_{i_3} and x_{i_1}, implying $f(x_{i_3}) = f(x_{i_1})$ as required.

Finally, we claim that the graph of the above construction is planar and has treewidth at most three. Planarity can be easily verified. We propose a tree decomposition as follows to establish the treewidth bound. The root of the tree has a bag containing v_0, v_1. The root has a child for each variable x_i, with a bag containing v_0, v_1, x_i. In addition, there is a two-node path connected to the root corresponding to each clause and its gadget: let a_i, b_i, c_i be the vertices of the gadget. Then, the root of the tree decomposition has a child, whose bag is v_0, v_1, a_i, b_i, and has a child of its own with a bag a_i, b_i, c_i. The largest bag has size four, and the endpoints of each edge of the graph appear together in at least one tree node. □

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for thoughtful comments that led to improved presentation of the paper. Special thanks goes to one reviewer for pointing out the runtime improvement of Theorem 5.1.

REFERENCES

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

Agrawal, A., Klein, P. N., and Ravi, R. 1991. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, USA, 134–144.

Archer, A., Bateni, M., Hajiaghayi, M., and Karloff, H. 2009. Improved approximation algorithms for prize-collecting Steiner tree and TSP. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Washington, DC, USA.

Arora, S. 1996. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Washington, DC, USA, 2.

Arora, S. 1998. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM 45,* 5, 753–782.

Baker, B. S. 1994. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM 41,* 1, 153–180.

Bateni, M., Chekuri, C., Ene, A., Hajiaghayi, M., Korula, N., and Marx, D. 2011. Prize-collecting Steiner problems on planar graphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, USA, 1028–1049.

Bateni, M., Hajiaghayi, M., Klein, P. N., and Mathieu, C. 2011. A polynomial-time approximation scheme for planar multiway cut. submitted.

Berman, P. and Ramaiyer, V. 1992. Improved approximations for the Steiner tree problem. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, USA, 325–334.

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

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

Borradaile, G., Demaine, E. D., and Tazari, S. 2009. Polynomial-time approximation schemes for subset-connectivity problems in bounded genus graphs. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS)*. Springer, New York, NY, USA, 171–182.

Borradaile, G., Klein, P. N., and Mathieu, C. 2008. A polynomial-time approximation scheme for Euclidean Steiner forest. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Washington, DC, USA, 115–124.

Borradaile, G., Klein, P. N., and Mathieu, C. 2009. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms 5,* 3.

Borradaile, G., Mathieu, C., and Klein, P. N. 2007. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, USA, 1285–1294.

Byrka, J., Grandoni, F., Rothvoss, T., and Sanità, L. 2010. An improved LP-based approximation for Steiner tree. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, USA, 583–592.

Cabello, S. and Chambers, E. W. 2007. Multiple source shortest paths in a genus g graph. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, USA, 89–97.

Cabello, S. and Mohar, B. 2005. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. In *Proceedings of the 13th Annual European Symposium of Algorithms (ESA)*. Springer, New York, NY, USA, 131–142.

Demaine, E. D., Hajiaghayi, M., and Mohar, B. 2007. Approximation algorithms via contraction decomposition. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, USA, 278–287.

Erickson, R. E., Monma, C. L., , and Veinott, A. F. 1987. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research 12*, 634–664.

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

Gassner, E. 2009. The Steiner forest problem revisited. *Journal of Discrete Algorithms In Press, Corrected Proof*, –.

Goemans, M. X. and Williamson, D. P. 1995. A general approximation technique for constrained forest problems. *SIAM Journal on Computing 24,* 2, 296–317.

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

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

Karpinski, M. and Zelikovsky, A. 1997. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization 1,* 1, 47–65.

Klein, P. N. 2008. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing 37,* 6, 1926–1952.

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

Marx, D. 2007. Precoloring extension on chordal graphs. In *Graph Theory in Paris. Proceedings of a Conference in Memory of Claude Berge*, A. Bondy, J. Fonlupt, J.-L. Fouquet, J.-C.

Fournier, and J. Ramirez Alfonsin, Eds. Trends in Mathematics. Birkhäuser, Basel, Switzerland, 255–270.

Marx, D. 2009. Complexity results for minimum sum edge coloring. *Discrete Applied Mathematics 157,* 5, 1034–1045.

Mitchell, J. S. B. 1999. 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.

Mohar, B. and Thomassen, C. 2001. *Graphs on surfaces.* Johns Hopkins University Press, Baltimore, MD.

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

Richey, M. B. and Parker, R. G. 1986. On multiple Steiner subgraph problems. *Networks 16,* 4, 423–438.

Robertson, N. and Seymour, P. D. 1986. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms 7,* 3, 309–322.

Robertson, N. and Seymour, P. D. 1994. Graph minors. XI. circuits on a surface. *Journal of Combinatorial Theory, Series B 60,* 1, 72–106.

Robins, G. and Zelikovsky, A. 2005. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics 19,* 1, 122–134.

Schaefer, T. J. 1978. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC).* ACM, New York, NY, USA, 216–226.

Thimm, M. 2003. On the approximability of the Steiner tree problem. *Theor. Comput. Sci. 295,* 1-3, 387–402.

Zelikovsky, A. 1992. An 11/6-approximation for the Steiner problem on graphs. In *Proceedings of the Fourth Czechoslovakian Symposium on Combinatorics, Graphs, and Complexity (1990) in Annals of Discrete Mathematics.* Vol. 51. Elsevier, 351–354.

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

Zelikovsky, A. 1996. Better approximation bounds for the network and Euclidean Steiner tree problems. Tech. rep., University of Virginia, Charlottesville, VA, USA.

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

Received Month Year; revised Month Year; accepted Month Year