Prize-collecting Steiner Problems on Planar Graphs

M. Bateni[*]   C. Chekuri[†]   A. Ene[‡]   M.T. Hajiaghayi[§]   N. Korula[¶]   D. Marx[‖]

Abstract

In this paper, we reduce Prize-Collecting Steiner TSP (PCTSP), Prize-Collecting Stroll (PCS), Prize-Collecting Steiner Tree (PCST), Prize-Collecting Steiner Forest (PCSF), and more generally Submodular Prize-Collecting Steiner Forest (SPCSF), on planar graphs (and also on bounded-genus graphs) to the corresponding problem on graphs of bounded treewidth. More precisely, for each of the mentioned problems, an α-approximation algorithm for the problem on graphs of bounded treewidth implies an $(\alpha + \epsilon)$-approximation algorithm for the problem on planar graphs (and also bounded-genus graphs), for any constant $\epsilon > 0$. PCS, PCTSP, and PCST can be solved exactly on graphs of bounded treewidth and hence we obtain a PTAS for these problems on planar graphs and bounded-genus graphs. In contrast, we show that PCSF is APX-hard to approximate on series-parallel graphs, which are planar graphs of treewidth at most 2. Apart from ruling out a PTAS for PCSF on planar graphs and bounded treewidth graphs, this result is also interesting since it gives the first provable hardness separation between the approximability of a problem and its prize-collecting version. We also show that PCSF is APX-hard on Euclidean instances.

1 Introduction

In this paper we consider prize-collecting versions of several network design problems. A typical network design problem is modeled as the problem of finding a minimum-cost sub-network of a given network G that satisfies some "requests". The requests often correspond to connectivity between some given pairs or sets of nodes. In prize-collecting variants, each request has a penalty, and we allow the sub-network not to satisfy some requests. The goal is to minimizie the cost of the sub-network plus the penalties for the requests that are not satisfied by the sub-network. These problems are interesting for several reasons. In particular, prize-collecting Steiner problems are well-known network design problems with several applications in expanding telecommunications networks (see for example [42, 49]), cost sharing, and Lagrangian relaxation techniques (see e.g. [41, 26]). A general problem in this area is the Prize-Collecting Steiner Forest (PCSF) problem[1]: given an undirected network (graph) $G = (V, E)$, a set of source-sink pairs[2] $\mathcal{D} = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_k, t_k\}\}$, a non-negative edge-cost function $c\colon E \to \mathbb{R}_+$, and a non-negative penalty function $\pi\colon \mathcal{D} \to \mathbb{R}_+$, the goal is to find a subgraph H of G to minimize the cost of the edges of H plus the sum of the penalties for requests in $\mathcal{D}$ that are not connected by H. A more general problem is obtained if the penalty for not connecting a set of demands is not simply the sum of individual penalties for unconnected demands, but an arbitrary function $\pi\colon 2^{\mathcal{D}} \to R_+$. A natural and useful restriction on π is that it is a monotone and non-negative *submodular* function[3]; in this case we obtain the Submodular Prize-Collecting Steiner Forest (SPCSF) of all unsatisfied pairs.

---

[1]In the literature, this problem is also called Prize-Collecting Generalized Steiner Tree.

[2]Source-sink pairs are sometimes called demands.

[3]A function $f : 2^S \mapsto \mathbb{R}$ is called *submodular* if and only if $\forall A, B \subseteq S : f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. An equivalent characterization is that the marginal profit of each item should be non-increasing, i.e., $f(A \cup \{a\}) - f(A) \leq f(B \cup \{a\}) - f(B)$ if $B \subseteq A \subseteq S$ and $a \in S \setminus B$. A function $f : 2^S \mapsto \mathbb{R}$ is *monotone* if and only if $f(A) \leq f(B)$ for $A \subseteq B \subseteq S$. Since the number of sets is exponential, we assume a value oracle access to the submodular function; i.e., for a given set T, an algorithm can query an oracle to find its value $f(T)$.

The prize-collecting problems generalize the underlying network design problems since one can set the penalties to ∞ which forces the solution to satisfy all requests. In particular, PCSF generalizes the well-studied Steiner Forest problem which is NP-Hard and also APX-Hard to approximate. The best known approximation ratio for Steiner Forest is $2 - \frac{2}{n}$ (n is the number of nodes of the graph) due to Agrawal, Klein, and Ravi [2] (see also [35] for a more general result and a simpler analysis). The case of Prize-Collecting Steiner Forest problem in which all sinks are identical is the (rooted) Prize-Collecting Steiner Tree (PCST) problem. In the unrooted version of this problem, there is no specific sink (root); here, the goal is to find a tree connecting some sources and pay the penalty for the rest of them. We also study two variants of (unrooted) Prize-Collecting Steiner Tree, Prize-collecting TSP (PCTSP) and Prize-collecting Stroll (PCS), in which the set of edges form a cycle and a path (respectively) instead of a tree. When in addition all penalties are ∞ in these prize-collecting problems, we have the classic APX-hard problems Steiner Tree, TSP and Stroll (Path TSP) for which the best approximation factors in order are 1.39 [19], $\frac{3}{2}$ [25], and $\frac{3}{2}$ [39].

Why are prize-collecting problems interesting? PCST and PCTSP are two classic optimization problems with a large impact, both in theory and practice. At AT&T, PCST code has been used in large-scale studies in access network design, both as described by Johnson, Minkoff and Phillips [42], and in another unpublished applied work by Archer et al..

The key difference between problems such as PCST, PCSF and their special cases Steiner Tree and Steiner Forest is that we do not know *a priori* the set of demands that are to be satisfied/connected; satisfying more demands reduces the penalty, but increases the connection cost. This connection cost plus penalty nature of the objective function models realistic problems with multiple goals; for example, in network construction, one may wish to examine the tradeoff between the cost of serving clients and the potential profit from serving them. The impact of PCST and PCTSP within approximation algorithms is also far-reaching, especially in the study of other problems where the set of demands to be satisfied is not fixed: In the k-MST and k-Stroll problems [30, 8, 27, 7, 31, 21], the goal is to find a minimum-cost tree or path containing at least k vertices, and in the Max-Prize-Tree and Orienteering problems [16, 11, 24, 46], the goal is to find a tree or path that contains as many vertices as possible, subject to a length constraint. In particular, PCST is a Lagrangian relaxation of the k-MST problem, and hence has played a crucial role in the design of algorithms for all the problems mentioned above. Thus, we are motivated to study

prize-collecting problems both for their inherent theoretical and practical value, and because they are useful in the study of several other problems of interest.

In this paper, we consider prize-collecting problems in planar graphs. Planarity is a natural restriction for network design in some practical scenarios such as telecommunication networks where crossings between cables or fiber in the ground are few in number if at all. Thus obtaining algorithms with better approximation factors is desirable in this case. There is a wealth of literature on obtaining improved approximation algorithms for planar graphs. Here we focus on PTASs. The seminal work of Baker [9] obtained PTASs for several optimization problems on planar graphs (such as minimum vertex cover and maximum independent set) although the corresponding problems on general graphs are considerably harder to approximate. The main idea in her work is a decomposition approach that reduces the problem on a planar graph to the problem on graphs of bounded treewidth. This approach has been subsequently applied in a variety of contexts. (The algorithmic and graph-theoretic properties of treewidth are extensively studied and a well-understood dynamic programming technique can solve NP-hard problems on bounded treewidth graphs.) The broad outline of the PTAS approach for planar graphs had to be augmented with a variety of non-trivial ideas and extensions. In this paper we consider prize-collecting network design problems. Before we discuss our contributions, we describe some prior work on network design problems in planar graphs.

TSP, Steiner Tree, and Steiner Forest all have been considered extensively on planar graphs. Indeed, all these problems remain NP-hard even in this setting [29]. However, obtaining a PTAS for each of these problems remained an open problem for several years. Grigni, Koutsoupias, and Papadimitriou [36] obtained the first PTAS for TSP on unweighted planar graphs in 1995; this was later generalized to weighted planar graphs [6] (and improved to linear time [44]). Obtaining a PTAS for Steiner Tree on planar graphs remained elusive for almost 12 years until 2007 when Borradaile, Klein and Mathieu [18] obtained the first PTAS for Steiner Tree on planar graphs using a new technique of contraction decomposition and building spanners (this borrowed ideas from earlier work of Klein on Subset TSP [43]) [18] posed obtaining a PTAS for Steiner Forest in planar graphs as the main open problem. Bateni, Hajiaghayi and Marx [13] very recently solved this open problem using a primal-dual technique for building spanners and obtaining PTASs by reducing the problem to bounded-treewidth graphs. Interestingly, Steiner Forest turns out to be NP-hard even on graphs of treewidth 3

and hence [13] had to devise a PTAS for the case of bounded treewidth graphs in order to apply the general framework.

Obtaining PTASs for prize-collecting versions of the above network design problems was suggested as an open problem in [13, 12]. The main technical difficulty in prize-collecting problems is that it is not apriori clear which requests are to be satisfied. In this paper, we resolve this difficulty for PCST, PCTSP, PCSF, and even more generally, for SPCSF, by reducing these problems on planar graphs to the corresponding problems on graphs of bounded treewidth. More precisely we show that any α-approximation algorithm for these problems on graphs of bounded treewidth gives an $(\alpha + \epsilon)$-approximation algorithm for these problems on planar graphs and bounded-genus graphs, for any constant $\epsilon > 0$. Since PCST and PCTSP can be solved exactly on graphs of bounded treewidth using standard dynamic programming techniques (as we discuss later in the paper), we immediately obtain PTASes for PCST and PCTSP on planar graphs (the same holds for PCS as well). In contrast, we show that PCSF is APX-hard already on series-parallel graphs, which are planar graphs of treewidth at most 2, ruling out a PTAS for planar PCSF (assuming $P \neq NP$). Apart from ruling out a PTAS for PCSF on planar graphs and bounded treewidth graphs, this result is also interesting since it gives the first provable hardness separation between the approximability of a problem and its prize-collecting version; in this case Steiner Forest and Prize-Collecting Steiner Forest when restricted to planar graphs. We also show that PCSF is APX-hard on Euclidean instances, that is, when the input graph is induced by points in the Euclidean plane and the lengths are Euclidean distances.

1.1   Related Work We have already mentioned several related papers; we discuss these and others below. As described above, PCST is a Lagrangian relaxation of the k-MST problem, and has been used in a sequence of papers ([30, 8, 27, 7]) culminating in a 2-approximation algorithm for k-MST by Garg [31]. PCTSP has also been used to improve the approximation ratio and running time of algorithms for the Minimum Latency problem ([5, 22]). The first approximation algorithms for PCST and PCTSP were given by Bienstock et al. [15], although PCTSP had been introduced earlier by Balas [10]. Bienstock et al. achieved a factor of 3 for PCST and 2.5 for PCTSP by rounding the optimal solution to a linear programming (LP) relaxation. Later, Goemans and Williamson [34] constructed primal-dual algorithms using the same LP relaxation to obtain a 2-approximation for both problems, building

on work of Agrawal, Klein and Ravi [2]. Chaudhuri et al. [22] modified the Goemans-Williamson algorithm to achieve a 2-approximation algorithm for PCS. It is only recently that this factor of 2 for PCST and PCTSP was improved by Archer et al. [4]; they obtained a ratio for 1.967 for PCST and 1.980 for PCTSP; Goemans [33] combined some ideas of [4] with others from [32] to improve the ratio for PCTSP below 1.915.

The Prize-Collecting Steiner Forest problem was first considered by Hajiaghayi and Jain [37]. The technique of Bienstock et al. [15] easily implies a 3-approximation but requires the solution to a primal LP. In contrast, [37] developed an improved 2.54 approximation via the LP, and a technically interesting 3-approximation via a sophisticated primal-dual algorithm. Their primal-dual approach has been generalized by Sharma, Swamy, and Williamson [50] to SPCSF and related problems.

2   Technical Contributions and Overview

We first formally define the most general problem studied in this paper. An instance of Submodular Prize-Collecting Steiner Forest (SPCSF) is described by a triple $(G, \mathcal{D}, \pi)$ where G is a undirected weighted graph, $\mathcal{D}$ is a set of $d_i = \{s_i, t_i\}$ demand pairs, and $\pi : 2^{\mathcal{D}} \mapsto \mathbb{R}^+$ is a monotone nonnegative submodular penalty function. A demand $d = \{s, t\}$ is *satisfied* by a subgraph F if and only if s, t are connected in F. If a forest F satisfies a subset $\mathcal{D}^{\mathsf{sat}}$ of the demands, its cost is defined as $\mathsf{cost}(F) := \mathsf{length}(F) + \pi(\mathcal{D}^{\mathsf{unsat}})$, where $\mathsf{length}(F)$ is a shorthand for the total length of all edges in F, and $\mathcal{D}^{\mathsf{unsat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{sat}}$ denotes the subset of unsatisfied demands.

We similarly define SPCTSP, SPCS and SPCST that are submodular prize-collecting variants of Traveling Salesman Problem, Stroll and Steiner Tree, respectively. An instance of these problems is represented by $(G, \mathcal{D}, \pi)$ where all the demands $d = \{s, t\} \in \mathcal{D}$ share a common root vertex $r \in V(G)$.[4] A feasible solution F is a TSP tour, stroll, or Steiner tree, respectively for a subset of demands, say $\mathcal{D}^{\mathsf{sat}} \subseteq \mathcal{D}$. The cost is then $\mathsf{cost}(F) := \mathsf{length}(F) + \pi(\mathcal{D}^{\mathsf{unsat}})$, where $\mathcal{D}^{\mathsf{unsat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{sat}}$.

We first show that SPCSF on planar graphs (or more generally, bounded-genus graphs) can be reduced in an approximation-preserving fashion (within a $(1+\epsilon)$-factor) to SPCSF on graphs of bounded-treewidth; refer to Appendix A for definitions regarding the treewidth and bounded-treewidth graphs as well as bounded-genus

---

[4]Both the rooted and unrooted variants of these problems may be more naturally defined with single-vertex demands rather than demand pairs; having such a formulation, we can guess one vertex of the solution, designate it as the root and obtain the rooted formulation as defined in this paper.

graphs. In the rest of the paper, we focus on planar graphs. The algorithms and analysis can be extended with minor modifications to work for bounded-genus graphs following prior ideas.

THEOREM 2.1. *For any given constant $\epsilon > 0$, an α-approximation algorithm for SPCSF on graphs of bounded treewidth implies a $(\alpha + \epsilon)$-approximation algorithm for SPCSF on planar graphs.*

The reduction of Theorem 2.1 involves three steps:

1. Given an instance of SPCSF, let OPT denote the cost of an optimal solution. We construct a collection of trees $\{\hat{T}_1, \ldots, \hat{T}_k\}$ with two important properties:

   (a) The total length of the trees is bounded; $\sum_i \mathsf{length}(\hat{T}_i) \leq f(\epsilon)\mathsf{OPT}$, for some function f depending only on ϵ.

   (b) Paying the penalty for all demand pairs not contained in the same tree does not significantly increase the cost of an optimal solution. More formally, let $\hat{\mathcal{D}}$ denote the set of demand pairs which are not both contained in the same tree. There is a solution F such that, if $\bar{\mathcal{D}}$ is the set of demands not satisfied by F, $\mathsf{length}(F) + \pi(\bar{\mathcal{D}} \cup \hat{\mathcal{D}}) \leq (1 + O(\epsilon))\mathsf{OPT}$.

2. Given the collection of trees, construct a *spanner* graph H, which is a subgraph of the input graph G with the following two properties: First, the total cost of edges in H is at most $f'(\epsilon)\mathsf{OPT}$, for some function f' depending only on ϵ. And second, there is a solution *contained in H* of cost $(1 + O(\epsilon))\mathsf{OPT}$. This follows the approach of Borradaile et al. [18, 13].

3. After constructing the spanner graph, we invoke a theorem implicit in the work of Klein [44] (reformulated by [28]) that allows us to pay a cost of at most $\epsilon\mathsf{OPT}$ while converting H to a graph of bounded treewidth. We then use the approximation algorithm to solve the instance of SPCSF on this bounded treewidth graph.

The second and third steps of the reduction are standard in recent works, and we focus our attention on the first step. Recall that the additional difficulty in solving PCST, PCSF, SPCSF, and related problems comes from not knowing which demands to connect. The first step implies that we can effectively focus our attention *only* on the demand pairs that have both vertices in the same tree. The core of the reduction, then, is obtaining the desired collection of trees, and

our algorithm is based on a *prize-collecting clustering* technique that was first implicitly used in [4] and later developed in [13]. In this work, the clustering technique is generalized as follows: First, we need to extend the ideas to work for prize-collecting variants of Steiner network problems. This can indeed make the problem provably harder; see Theorem 2.3. The original prize-collecting clustering associates a potential value to each node and grows the corresponding clusters consuming these potentials. However, in order to extend it to the prize-collecting setting, we consider source-sink potentials. This means that there is some interaction between the potentials of different nodes. Secondly, we consider submodular penalty functions that model even more interaction between the demands. The extended prize-collecting clustering procedure has two phases. In the first phase, we have a source-sink moat-growing algorithm, and in the second phase, we have a single-node potential moat-growing like [13].

Section 3 is devoted to the formal proof of Theorem 2.1. The algorithm starts with a constant-approximate solution F^1, say, obtained using Haji-aghayi et al. [38] who prove a 3-approximation for SPCSF on general graphs. The forest F^1 satisfies a subset of demands, and we know the total penalty of unsatisfied demands is bounded. The algorithm then tries to satisfy more demands by constructing a forest $F^2 \supseteq F^1$ whose length is bounded; see RestrictDemands in Section 3.4. This step heavily uses a *submodular prize-collecting clustering* algorithm[5] introduced in Section 3.3. At the end of this step, we can assume that the near-optimal solution does not satisfy the demands which are unsatisfied in F^2. Submodularity poses several difficulties in proving this property: ideally, we want to say that the cost paid by the optimal solution to satisfy these demands is significantly more than their penalty value. Surprisingly, this is not true. Nevertheless, we can prove that the *marginal cost* of the demands satisfied in the near-optimal solution but not in F^2 can be charged to the cost the near-optimal solution pays in order to satisfy them. The next step of the reduction is to build a forest $F^3 \supseteq F^2$ of bounded length that may connect several components of F^2 together; see Section 3.5. This is done by assigning to each component of F^2 a potential proportional to its length, and then running a prize-collecting clustering similar to that

---

[5]The algorithm bears some similarity to the primal-dual moat-growing algorithms for the Steiner network problems. One key difference is that we do not have a primal LP. We have an LP similar to the dual linear programs used in such algorithms, and we use a notion of potential as a substitute for the lack of the primal LP. The potentials, among other things, play the role of an upper bound for the value of the dual LP.

of [13]. This guarantees that the near-optimal solution does not need to connect different components of F^3 to each other.

Once we have the forest F^3 with components that do not need to be connected, we can implement Step 2 of our reduction: We construct a spanner (see [13, 18, 44]) out of each component of F^3 separately from the others. In the previous work [13], we could solve each of the subinstances independently, however, the penalty interaction originating from the submodular penalty function in the current work does not allow us to solve each subinstance completely independently. Instead, we say that the forest of the near-optimal solution on each subinstance is independent of the others.

Finally, after constructing the spanner graph F^4, we invoke a generalization of the shifting idea of Baker [9] due to [44, 28], and end up with a graph of bounded treewidth. Since bounded-treewidth graphs bear some similarity to trees, several tools have been developed for solving optimization problems on them. Standard techniques, see Appendix B, allow us to obtain PTASs for several Steiner network problems on graphs of bounded treewidth.

THEOREM 2.2. *PCST, PCS and PCTSP admit PTASs on bounded-treewidth graphs.*

In Section 4 we show how this results in PTASs for the above problems on planar graphs. In particular, this is simple for PCST since it is a special case of SPCSF. For the other two problems, however, refer to the discussion in Section 4.

In contrast, we show Prize-Collecting Steiner Forest is APX-hard, even on planar graphs of treewidth at least two; Hajiaghayi and Jain show the problem can be solved in polynomial on tree metrics [37].

THEOREM 2.3. *PCSF is APX-hard on (1) planar graphs of treewidth two and on (2) the two-dimensional Euclidean metric.*

This is done via a reduction from Bounded-Degree Vertex Cover in Section 5. Indeed, the result shows that Submodular Prize-Collecting Steiner Tree (the version of the problem when the solution has to be a connected tree instead of a forest) is also APX-hard. This implies the hardness of PCSF originates from the interaction between the penalties of terminals rather than from the different components of the solution.

Surprisingly, the hardness also works for Euclidean metrics, answering an open question raised in [12]. This is a very rare instance where a natural network optimization problem is APX-hard on the two-dimensional Euclidean plane.

Theorem 2.3 means that planar PCSF reaches a level of complexity where even though reduction to bounded-treewidth instances works, it does not give us a PTAS for the problem (in fact, no PTAS exists unless P = NP). However, the treewidth reduction approach can be still useful for obtaining constant-factor approximations for planar graphs better than the factor 2.54 algorithm of [37] for general graphs. Theorem 2.1 show that beating the 2.54 factor on bounded-treewidth graphs would immediately imply the same for planar graphs. We pose it as an open question whether this is indeed possible for PCSF.

Remarks: The current paper combines results obtained in independent papers of Bateni, Hajiaghayi and Marx [14] and Chekuri, Ene and Korula [23]. Although [14] was done slightly before [23], the authors of the latter work were not aware of the former before obtaining their results. We briefly describe the contributions of each work. The paper of Chekuri et al. gives a reduction from PCST, PCSF, PCTSP, and PCS on planar graphs to the corresponding problems on graphs of bounded treewidth. The reduction (see Section 3.1 for a special case) relies on properties of a primal-dual algorithm for the underlying problem with scaled up penalties. The reduction outlined by Bateni et al. works for the more general PCSF. Bateni et al. (see Section 3.2) use a separate primal-dual clustering step on top of the trees returned by an approximation algorithm (used as a black box) for the underlying problem, which is inspired by earlier work of Archer et al. [4] and further extended in [13]. The APX-hardness proofs are due to Bateni et al.; see Section 5. This paper mostly follows [14] with Section 3.1 based on the work in [23].

3   Reduction to the bounded-treewidth case

This section focuses on proving Theorem 2.1. In fact, we prove a stronger version of the theorem that is necessary for obtaining PTASs for PCST, PCTSP, and PCS. We reduce an instance $(G, \mathcal{D}, \pi)$ of SPCSF to an instance $(H, \mathcal{D}, \pi')$ where H has bounded treewidth and π' has a structure similar to π; in particular, for some $\mathcal{D}^{\mathsf{unsat}} \subseteq \mathcal{D}$ we define $\pi'(D) := \pi(D \cup \mathcal{D}^{\mathsf{unsat}})$ for all $D \subseteq \mathcal{D}$. Notice that if π is submodular, then so is π'. Moreover, if π models a PCSF instance, i.e., π is an additive function, then $\pi'(D) - \pi'(\emptyset)$ models a PCSF instance too. In fact, $\pi'(D)$ is an additive function that is shifted by a fixed amount $\pi'(\emptyset)$. The same condition holds for PCST, PCTSP and PCS. Therefore, after reducing a PCST instance, we are left with a PCST instance—rather than an SPCSF one—on a bounded-treewidth graph.

Before presenting the proof of the general reduction, we present in Section 3.1 a simpler proof that suffices

to obtain the desired reduction for PCST, PCTSP, PCS, and (with additional work) PCSF. However, this technique does not suffice to obtain the reduction for SPCSF. For ease of exposition, we focus on PCST in Section 3.1.

3.1 A Simpler Reduction for PCST Recall that our reduction to bounded-treewidth instances involved three steps; in this section, we omit discussions of the latter two (see the proof of Theorem 2.1 at the end of Section 3.2). The first step in the reduction is to find a collection of trees with the following two properties: First, their total length is $f(\epsilon)$OPT, and second, there is a solution to the SPCSF instance of cost at most $(1+\epsilon)$OPT that only connects demand pairs in the same tree.

For PCST, all demand pairs involve a common root; we construct a *single* tree $\hat{T}$ of length $O(1/\epsilon)$OPT that captures "almost all" of the crucial vertices: Even if we pay the penalty for all vertices not in $\hat{T}$, this does not significantly increase the cost of an optimal solution. More formally, we find a tree $\hat{T}$ of length $O(1/\epsilon)$OPT such that there exists a tree T with $\mathsf{length}(T) + \sum_{v \notin T \cap \hat{T}} \pi(v) \le (1 + \epsilon)$OPT. In fact, we can construct such a tree $\hat{T}$ that captures almost all the vertices of *any* optimal solution. We devote the rest of this section to describing the construction of this tree.

Given an instance I of PCST on a graph $G(V, E)$, with non-negative edge-cost function c and with $\pi(v)$ the penalty for not connecting vertex v to the root, we define a new instance I' as follows: The graph and edge-cost functions are unchanged, but we scale the penalties so that the penalty for not connecting v to the root is $\pi'(v) = \pi(v)/\epsilon$.

We now run the 2-approximate primal-dual algorithm `GW-Primal-Dual` of Goemans and Williamson [35] on the PCST instance I'. This algorithm is based on the following primal and dual linear programming formulations for PCST. For each vertex v, the variable z_v is 1 if we pay the penalty for not connecting v to the root r, and 0 otherwise; the variable x_e denotes whether the edge e is selected for the forest. Let $\mathcal{S}_v$ denote the collection of sets S that contain v but not r.

Primal-PCST

$$\min \sum_e c(e)x_e + \sum_v \pi(v)z_v$$
$$\sum_{e \in \delta(S)} x_e \ge (1 - z_v) \qquad (\forall i, S \in \mathcal{S}_v)$$
$$x_e, z_v \ge 0 \qquad (\forall e, v)$$

Dual-PCST

$$\max \sum_v \sum_{S \in \mathcal{S}_v} y_{v,S}$$
$$\sum_{S:\, e \in \delta(S)} \sum_{v:\, S \in \mathcal{S}_v} y_{v,S} \le c(e) \qquad (\forall e)$$
$$\sum_{S \in \mathcal{S}_v} y_{v,S} \le \pi(v) \qquad (\forall i)$$
$$y_{v,S} \ge 0 \qquad (\forall i, S \in \mathcal{S}_v)$$

Due to space constraints, we do not describe the well-known algorithm `GW-Primal-Dual` here, but observe that it returns both a tree $\hat{T}$ and a feasible dual solution with variables $y_{v,S}$, such that for all $v \notin \hat{T}$, $\sum_{S \in \mathcal{S}_v} y_{v,S} = \pi(v)$.

THEOREM 3.1. *Let T^* be any optimal solution to an instance I of PCST, and let* OPT $= \sum_{e \in T^*} c(e) + \sum_{v \notin T^*} \pi(v)$. *Let $\hat{T}$ be the tree output by algorithm* `GW-Primal-Dual` *on the instance I' with penalties scaled as above. Let X denote the set of vertices in T^* but not in $\hat{T}$. Then, $\sum_{e \in \hat{T}} c(e) \le 2OPT/\epsilon$, and $\sum_{v \in X} \pi(v) \le \epsilon$OPT.*

The tree $\hat{T}$ described in the theorem above satisfies the two properties we desire: Its length is comparable to OPT, and paying the penalty for all vertices not in $\hat{T}$ increases the cost of an optimal solution to the instance I by at most ϵOPT. To see the latter fact, fix an optimal solution T^*; by definition $\mathsf{length}(T^*) + \sum_{v \notin T^*} \pi(v) =$ OPT. But $\sum_{v \in T^* - \hat{T}} \pi(v) \le \epsilon$OPT by the theorem, and so $\mathsf{length}(T^*) + \sum_{v \notin T^* \cap \hat{T}} \pi(v) \le (1 + \epsilon)$OPT.

Hence, to complete the first step of the reduction, it suffices to prove Theorem 3.1.

Proof. This requires showing that the length of $\hat{T}$ is bounded $(\sum_{e \in \hat{T}} c(e) \le 2OPT/\epsilon)$, and that we can afford to pay the penalty for vertices in T^* but not in $\hat{T}$ (that is, $\sum_{v \in X} \pi(v) \le \epsilon$OPT). To see the former condition is true, note that T^* is a feasible solution to instance I', and has cost at most $\mathsf{length}(T^*) + \sum_{v \notin T^*} \pi'(v) = \mathsf{length}(T^*) + \frac{1}{\epsilon} \sum_{v \notin T^*} \pi(v) \le \frac{1}{\epsilon} \left(\mathsf{length}(T^*) + \sum_{v \notin T^*} \pi(v) \right) = \frac{1}{\epsilon}$OPT. Hence, the cost of an optimal solution to I' is at most $(1/\epsilon)$OPT, and as $\hat{T}$ is a 2-approximate solution to I', it has cost at most $(2/\epsilon)$OPT.

It remains only to prove that the total penalty of vertices in X is small. Consider a Steiner Tree instance defined on these vertices: As T^* connects all the vertices in X to the root, the cost of an optimal Steiner tree for X is at most OPT. Suppose, by way of contradiction, that $\sum_{v \in X} \pi(v) > \epsilon$OPT, and hence that

$\sum_{v \in X} \pi'(v) > \mathsf{OPT}$. Now consider the following dual of a natural LP for the Steiner Tree instance induced by X:

$$\textbf{Dual-Steiner Tree}(X)$$

$$\max \sum_{S \text{ separating some } v \in X \text{ from } r} z_S$$

$$\sum_{S:\, e \in \delta(S)} z_S \leq c(e) \qquad (\forall e)$$

$$z_S \geq 0 \qquad (\forall S)$$

Let $y_{v,S}$ be the feasible solution to **Dual-PCST** returned by `GW-Primal-Dual` on instance I'. Now, construct a dual solution to the LP **Dual-Steiner Tree**(X) as follows: For each set S separating some $v \in X$ from the root, set $z_S = \sum_{v \in X} y_{v,S}$. As $\sum_{S:e \in \delta(S)} \sum_{v:\, S \in \mathcal{S}_v} y_{v,S} \leq c(e)$ from the feasibility of the solution to **Dual-PCST**, we conclude that the dual variables z_S correspond to a feasible solution of **Dual-Steiner Tree**(X).

Thus, we have a feasible solution to **Dual-Steiner Tree**(X) of total value $\sum_S \sum_{v \in X:\, S \in \mathcal{S}_v} y_{v,S}$. But the dual solution returned by `GW-Primal-Dual` has the property that for each $v \notin \hat{T}$ (and hence for each $v \in X$), $\sum_{S \in \mathcal{S}_v} y_{v,S} = \pi'(v)$. Therefore, we have a feasible solution to **Dual-Steiner Tree**(X) of total value $\sum_{v \in X} \pi'(v) > \mathsf{OPT}$. By weak duality, the length of any Steiner tree for X must also be greater than OPT. But T^* is a Steiner tree for X of total length at most OPT, which is a contradiction.

3.2 A General Reduction We now return to the more general reduction. Our proof has three steps:

1. We start with an instance $(G, \mathcal{D}, \pi)$ of SPCSF. We first take out a subset, say $\mathcal{D}^{\mathsf{unsat}}$, of demands whose cost of satisfying is too much compared to their penalties. Thus, we can focus on the remaining demands, say $\mathcal{D}^{\mathsf{sat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{unsat}}$.

2. Afterwards, we partition the remaining demands $\mathcal{D}^{\mathsf{sat}}$ into $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_p$ such that, roughly speaking, SPCSF can be solved separately on each of the demand sets without increasing the total cost substantially.

3. Finally, we build a *spanner* for each demand set $\mathcal{D}_i$, and use similar ideas as in [13] to reduce the problem to bounded-treewidth graphs.

The first step is carried out in the following theorem. The proof appears in Section 3.4, and uses a submodular prize-collecting clustering technique introduced in Section 3.3. This step allows us to focus on only a subset $\mathcal{D}^{\mathsf{sat}}$ of demands, and ignore the rest of the demands. The additional cost due to this is only $\epsilon \mathsf{OPT}$.

THEOREM 3.2. *Given an instance $(G, \mathcal{D}, \pi)$ of SPCSF (or SPCTSP or SPCS) and a parameter $\epsilon > 0$, we can construct in polynomial time a subgraph F of G, satisfying only a subset $\mathcal{D}^{\mathsf{sat}} \subseteq \mathcal{D}$ of demands, in effect leaving $\mathcal{D}^{\mathsf{unsat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{sat}}$ unsatisfied, such that*

1. $\mathsf{length}(F) \leq (6\epsilon^{-1} + 3)\mathsf{OPT}$, *and*

2. *the optimum of $(G, \mathcal{D}^{\mathsf{sat}}, \pi')$ is at most $(1 + \epsilon)\mathsf{OPT}$ where $\pi'(D) := \pi(D \cup \mathcal{D}^{\mathsf{unsat}})$ is defined for $D \subseteq \mathcal{D}^{\mathsf{sat}}$.*

At this point, we have a constant-approximate solution satisfying all the (remaining) demands. The second step is a generalization and extension of the work in [13]. We are trying to break the instance into smaller pieces. The solution to each piece is almost independent of the others, i.e., there is little interaction between them. The following theorem is proved in Section 3.5.

THEOREM 3.3. *Given an instance $(G, \mathcal{D}, \pi)$ of SPCSF, a forest F satisfying all the demands, and a parameter $\epsilon > 0$, we can compute in polynomial time a set of trees $\{\hat{T}_1, \ldots, \hat{T}_k\}$, 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. *The tree $\hat{T}_i$ spans all the terminals in $\mathcal{D}_i$.*

3. *The total length of the trees $\hat{T}_i$ is within a constant factor of the length of F, i.e., $\sum_{i=1}^k \mathsf{length}(\hat{T}_i) \leq (\frac{2}{\epsilon} + 1)\mathsf{length}(F)$.*

4. *Let $\mathcal{D}^*$ be the subset of demands satisfied by OPT. Define $\mathcal{D}_i^* := \mathcal{D}^* \cap \mathcal{D}_i$, and denote by $\mathsf{SteinerForest}(G, \mathcal{D})$ the length of a minimum Steiner forest of G satisfying the demands $\mathcal{D}$. We have $\sum_i \mathsf{SteinerForest}(G, \mathcal{D}_i^*) \leq (1 + \epsilon)\mathsf{SteinerForest}(G, \mathcal{D}^*)$.*

The final step is very similar to the spanner construction of [13, 18]. Since it has been extensively covered in those works, we defer the details to the full version of the paper[6].

Now we show how the above theorems imply the main theorem of the paper.

_____
[6]The previous work show for Steiner tree and Steiner forest that, given a subgraph of length $O(\mathsf{OPT})$ with sufficient connectivity as that of a near-optimal solution, we can construct a *spanner*, i.e., a subgraph such that (1) the total length of the subgraph is no more than a constant times the length of the cost of the optimal solution, and (2) there is a near-optimal (i.e., $(1 + \epsilon)$-approximate) solution using only the edges of the subgraph.

Proof. [Proof of Theorem 2.1] Start with an instance $(G, \mathcal{D}, \pi)$ of SPCSF. Without loss of generality we present an approximation guarantee of $\alpha + O(1)\epsilon$. Find F, $\mathcal{D}^{\mathsf{sat}}$ and $\mathcal{D}^{\mathsf{unsat}}$ from applying Theorem 3.2 on $(G, \mathcal{D}, \pi)$. We know that F satisfies $\mathcal{D}^{\mathsf{sat}}$ and $\mathsf{length}(F) = O(\mathsf{OPT})$. Moreover, $\mathsf{OPT}_{\mathcal{D}^{\mathsf{sat}}}(G) \leq \mathsf{OPT}$. Define $\pi^+(D) := \pi(D \cup \mathcal{D}^{\mathsf{unsat}})$ for all $D \subseteq \mathcal{D}$. Clearly, the optimal solution of $(G, \mathcal{D}^{\mathsf{sat}}, \pi^+)$ costs no more than $(1 + \epsilon)\mathsf{OPT}$. Pick $\epsilon' < \epsilon \cdot \mathsf{length}(F)/\mathsf{OPT}$ and feed $(G, \mathcal{D}^{\mathsf{sat}}, \pi^+)$ along with F and ϵ' into Theorem 3.3 in order to obtain $\mathcal{D}_i$'s and $\hat{T}_i$'s for $i = 1, \ldots, k$. We have $\sum_i \mathsf{length}(\hat{T}_i) = O(\mathsf{length}(F)) = O(\mathsf{OPT})$, since ϵ' is a constant. In addition, the theorem guarantees a near-optimal solution OPT^+ of cost at most $(1 + 2\epsilon)\mathsf{OPT}$ that does not use the connectivity of different components $\mathcal{D}_i$ and $\mathcal{D}_{i'}$ for $i, i' \in \{1, \ldots, k\} : i \neq i'$. This ensures that the spanner construction gives us a graph G^+ (of total length $O(\mathsf{OPT})$) that approximates the forest of the solution within a $1 + \epsilon$ factor. Thus, the optimal solution of $(G^+, \mathcal{D}^{\mathsf{sat}}, \pi^+)$ costs at most $(1 + \epsilon)(1 + 2\epsilon)\mathsf{OPT} = [1 + O(1)\epsilon]\mathsf{OPT}$. Since the total length of the graph G^+ is within $O(\mathsf{OPT})$, we can use the decomposition theorem[7] due to Klein [44] to reduce the problem to bounded-treewidth graphs with an increase of $\epsilon\mathsf{OPT}$ in the solution cost. The reduced instance is solved via the α-approximation algorithm, and we finally get an approximation ratio of $\alpha + O(\epsilon)$.

3.3 Submodular prize-collecting clustering
First we present and analyze a primal-dual algorithm for SPCSF, and later we see how this algorithm can be used to achieve the goal of identifying and removing certain demands from the optimal solution such that the additional penalty is negligible.

Consider an instance $(G(V, E), \mathcal{D}, \pi)$ of the SPCSF. A set $S \subseteq V$ is said to *cut* a demand $d = \{s, t\}$ if and only if $|S \cap d| = 1$. We denote this by the short-hand $d \odot S$, and say the demand d *crosses* the set S. In the linear program (3.1)–(3.1), there is a variable $y_{S,d}$ for any $S \subseteq V$, $d \in \mathcal{D}$ such that $d \odot S$. For convenience, we use the short-hands $y_S := \sum_{d \in \mathcal{D}} y_{S,d}$ and $y_d := \sum_{S \subseteq V} y_{S,d}$.

$$\sum_{S : e \in \delta(S)} y_S \leq c_e \qquad \forall e \in E$$

$$\sum_{d \in D} y_d \leq \pi(D) \qquad \forall D \subseteq \mathcal{D}$$

$$y_{S,d} \geq 0 \qquad \forall d \in \mathcal{D}, S \subseteq V, d \odot S.$$

We produce a solution to the above LP. Theorem 3.2

[7]This technique is first implicitly used in the conference version of Klein [44], and is subsequently reformulated in [28].

is proved via some properties of this solution. These constraints look like the dual of a natural linear program for SPCSF. For convenience, we use the notation $y(D) := \sum_{d \in D} y_d$ for any $D \subseteq \mathcal{D}$.

LEMMA 3.1. *Given an instance $(G, \mathcal{D}, \pi)$ of SPCSF, we produce in polynomial time a forest F and a subset $\mathcal{D}^{\mathsf{unsat}} \subseteq \mathcal{D}$ of demands, along with a feasible vector y for the above LP such that*

1. $y(\mathcal{D}^{\mathsf{unsat}}) = \pi(\mathcal{D}^{\mathsf{unsat}})$;

2. F satisfies any demand in $\mathcal{D}^{\mathsf{sat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{unsat}}$; and

3. $\mathsf{length}(F) \leq 2y(\mathcal{D})$.

The solution is built up in two stages. First we perform an *submodular growth* to find a forest F_1 and a corresponding y vector. This differs from the usual growth phase of [35, 1] in that the penalty function may go tight for a set of vertices that are not currently connected. In the second stage, we *prune* some edges of F_1 to obtain another forest F_2. Below we describe the two phases of Algorithm 1 (`Submodular-PC-Clustering`).

Growth We begin with a zero vector y, and an empty set F_1. A demand $d \in \mathcal{D}$ is said to be *live* if and only if $y(D) < \pi(D)$ for any $D \subseteq \mathcal{D}$ that $d \in D$. If a demand is not live, it is *dead*. During the execution of the algorithm `Submodular-PC-Clustering`, 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 there is a live demand $d : d \odot C$. We simultaneously *grow* all the active clusters by η. In particular, if there are $\kappa(C) > 0$ live demands crossing an active cluster C, we increase $y_{C,d}$ by $\eta/\kappa(C)$ for each live demand $d : d \odot C$. Hence, y_C is increased by η for every active cluster C. We pick the largest value for η that does not violate any of the constraints in (3.1) or (3.1). Obviously, η is finite in each iteration because the values of these variables cannot be larger than $\pi(\mathcal{D})$. Hence at least one such constraint goes tight after each 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 which 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 (3.1) becomes tight. The number of iterations is at most $2|V|$ because at each event either a demand dies, or the size of $\mathcal{C}$ decreases.

Computing η is nontrivial here. In particular, we have to solve an auxiliary linear program to find its value. New variables $y_{S,d}^*$ denote the value of vector y after a growth of size η. All the constraints are written

for the new variables. There are exponentially many constraints in this LP, however, it admits a separation oracle and thus can be optimized.[8]

$$\text{maximize } \eta$$

subject to

$$y^*_{S,d} = y_{S,d} + \frac{\eta}{\kappa(S)}$$
$$\forall d \in \mathcal{D}, S \subseteq V, d \odot S, \kappa(S) > 0$$

$$y^*_{S,d} = y_{S,d}$$
$$\forall d \in \mathcal{D}, S \subseteq V, d \odot S, \kappa(S) = 0$$

$$\sum_{S:e \in \delta(S)} y^*_S \leq c_e \qquad \forall e \in E$$

$$\sum_{d \in D} y^*_d \leq \pi(D) \qquad \forall D \subseteq \mathcal{D}$$

$$y^*_{S,d} \geq 0 \qquad \forall d \in \mathcal{D}, S \subseteq V, d \odot S.$$

Pruning Let $\mathcal{S}$ denote the set of all clusters formed 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 clusters C that do not cut any live demand. Notice that a demand d may still be live at the end of the growth stage if it is satisfied; roughly speaking, the demand is satisfied before it exhausts its *potential*. In the pruning stage, we iteratively remove edges from F_1 to obtain F_2. More specifically, we first initialize F_2 with F_1. Then, as long as there is a cluster $S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$, we remove the edge e from F_2.

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

We first bound the length of the forest F. The following lemma is similar to the analysis of the algorithm in [35]. However, we do not have a primal LP to give a bound on the dual. Rather, the upper bound for the length is $\pi(\mathcal{D})$. In addition, we bound the cost of a forest F that may have more than one connected compo-

---

Algorithm 1 Submodular-PC-Clustering

Input: Instance $(G(V,E), \mathcal{D}, \pi)$ of Generalized prize-collecting Steiner forest

Output: Forest F, subset of demands $\mathcal{D}^{\mathsf{unsat}}$ and fractional solution y.

1: Let $F_1 \leftarrow \emptyset$.
2: Let $y_{S,d} \leftarrow 0$ for any $d \in \mathcal{D}, S \subseteq V, d \odot S$.
3: Let $\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V^*\}$.
4: **while** there is a live demand **do**
5:     Compute η via LP (3.1): the largest possible value such that simultaneously increasing y_C by η for all active clusters $C \in \mathcal{C}$ does not violate Constraints (3.1)-(3.1).
6:     Let $y_{C,d} \leftarrow y_{C,d} + \frac{\eta}{\kappa(C)}$ for all live demands d crossing clusters $C \in \mathcal{C}$, i.e., $d \odot C$.
7:     **if** $\exists e \in E$ that is tight and connects two clusters C_1 and C_2 **then**
8:         Pick one such edge $e = (u, v)$.
9:         Let $F_1 \leftarrow F_1 \cup \{e\}$.
10:       Let $C \leftarrow C_1 \cup C_2$.
11:       Let $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{C_1, C_2\}$.
12:       Let $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$.
13: Let $F_2 \leftarrow F_1$.
14: Let $\mathcal{B}$ be the set of all clusters $S \in \mathcal{S}$ that do not cut any live demands.
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: Let $\mathcal{D}^{\mathsf{unsat}}$ denote the set of dead demands.
18: Output $F := F_2$, $\mathcal{D}^{\mathsf{unsat}}$ and y.

---

[8] Notice that there are only a polynomial number of non-zero variables at each step since $y_{S,d}$ may be non-zero only for clusters S, and these clusters form a laminar family in our algorithm. Verifying constraints (3.1)-(3.1) and (3.1) is very simple. Verifying constraints (3.1) is equivalent to finding $\min_{D \subseteq \mathcal{D}} \pi(D) - y^*(D)$ and checking that it is non-negative. The function to minimize is submodular and thus can be minimized in polynomial time [40]. A standard argument shows that the values of these variables have polynomial size. We defer to the full version of the paper the detailed discussion of how the LP can be approximated.

nent, whereas the prize-collecting Steiner tree algorithm of [35] finds a connected graph at the end.

LEMMA 3.2. *The cost of F_2 is at most $2y(\mathcal{D})$.*

Proof. 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 jth event point occurs in the growth phase ($0 = t_0 \leq t_1 \leq t_2 \leq \cdots$), so the jth 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 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 (3.1) became tight, we can exactly apportion the cost c_e amongst the collection of clusters $\{C : e \in \delta(C)\}$ whose variables "pay for" the edge, and can divide this up further by epoch. In other words, $c_e = \sum_j \sum_{C:e\in\delta(C)} y_C^{(j)}$. We will now prove that the total edge cost from F_2 that is apportioned to epoch j is at most $2\sum_C y_C^{(j)}$. In other words, during each epoch, the total rate at which edges of F_2 are paid for by all active clusters is at most twice the number of active clusters. Summing over the epochs yields the desired conclusion.

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

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 live clusters 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.

Now we can prove Lemma 3.1 that characterizes the output of `Submodular-PC-Clustering`.

Proof. [Proof of Lemma 3.1] For every demand $d \in \mathcal{D}^{\text{unsat}}$ we have a set $D \ni d$ such that $y(D) = \pi(D)$. The definition of $\mathcal{D}^{\text{unsat}}$ guarantees $D \subseteq \mathcal{D}^{\text{unsat}}$. Therefore, we have sets $D_1, D_2, \ldots, D_l$ that are all tight (i.e., $y(D_i) = \pi(D_i)$) and they span $\mathcal{D}^{\text{unsat}}$ (i.e., $\mathcal{D}^{\text{unsat}} = \cup_i D_i$). To prove $y(\mathcal{D}^{\text{unsat}}) = \pi(\mathcal{D}^{\text{unsat}})$, we use induction and combine D_i's two at a time. For any two tight sets A and B we have $y(A\cup B) = y(A) + y(B) - y(A\cap B) = \pi(A) + \pi(B) - y(A\cap B) \geq \pi(A) + \pi(B) - \pi(A\cap B) \geq \pi(A\cup B)$, where the second equation follows from tightness of A and B, the third step is a result of Constraint (3.1), and the last step follows from submodularity. Constraint (3.1) has it that $\pi(A\cup B) \geq y(A\cup B)$, therefore, it has to hold with equality.

Clearly, at the end of execution of `Submodular-PC-Clustering`, any live demand is already satisfied. Notice that such demands are not affected in the pruning stage. Hence, only dead demands may be not satisfied. This guarantees the second condition. The third condition follows from Lemma 3.2.

3.4 Restricting the demands We prove Theorem 3.2 in this section. First, we obtain a constant-factor approximate solution F^+—via the 3-approximation algorithm for general graphs [38]. Let $\mathcal{D}^+$ denote the demands satisfied by F^+. We denote by T_j^+ the connected components of F^+. For each demand $d = \{s,t\} \in \mathcal{D}^+$ we clearly have $\{s,t\} \subseteq V(T_j^+)$ for some j. However, for an unsatisfied demand $d' = \{s',t'\} \in \mathcal{D} \setminus \mathcal{D}^+$, the vertices s' and t' belong to two different components of F^+. Construct G^* from G by reducing the length of edges of F^+ to zero. The new penalty function π^* is defined as follows:

$$(3.1) \qquad \pi^*(D) := \epsilon^{-1}\pi(D) \qquad \text{for } D \subseteq \mathcal{D}.$$

Finally we run `Submodular-PC-Clustering` on $(G^*, \mathcal{D}, \pi^*)$; see Algorithm 2.

Now we show that the algorithm `Restrict-Demands` outlined above satisfies the requirements of Theorem 3.2. Before doing so, we show how the cost of a forest can be compared to the values of the output vector y.

LEMMA 3.3. *If a graph F satisfies a set $\mathcal{D}^{\text{sat}}$ of demands, then $\mathsf{length}(F) \geq \sum_{d\in\mathcal{D}^{\text{sat}}} y_d$.*

This is quite intuitive. Recall that the y variables color the edges of the graph. Consider a segment on

---
Algorithm 2 `Restrict-Demands`

Input: Instance $(G, \mathcal{D}, \pi)$ of Submodular Prize-Collecting Steiner Forest

Output: Forest F and $\mathcal{D}^{\mathsf{unsat}}$.

1: Use the algorithm of Hajiaghayi et al. [38] to find a 3-approximate solution: a forest F^+ satisfying subset $\mathcal{D}^+$ of demands.
2: Construct $G^*(V, E^*)$ in which E^* is the same as E except that the edges of F^+ have length zero in E^*.

3: Define π^* as Equation (3.1).
4: Call `Submodular-PC-Clustering` on $(G^*, \mathcal{D}, \pi^*)$ to obtain the result F, $\mathcal{D}^{\mathsf{unsat}}$ and y.
5: Output F and $\mathcal{D}^{\mathsf{unsat}}$.

---

edges corresponding to cluster S with color d. At least one edge of F *passes through* the cut $(S, \bar{S})$. Thus, a portion of the cost of F can be charged to $y_{S,d}$. Hence, the total cost of the graph F is at least as large as the total amount of colors paid for by $\mathcal{D}^{\mathsf{sat}}$. We now provide a formal proof.

Proof. The length of the graph F is

$$
\sum_{e \in F} c_e \geq \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S \qquad \text{by (3.1)}
$$

$$
= \sum_{S} |F \cap \delta(S)| y_S
$$

$$
\geq \sum_{S: F \cap \delta(S) \neq \emptyset} y_S
$$

$$
= \sum_{S: F \cap \delta(S) \neq \emptyset} \sum_{d: d \odot S} y_{S,d}
$$

$$
= \sum_{d} \sum_{\substack{S: d \odot S \\ F \cap \delta(S) \neq \emptyset}} y_{S,d}
$$

$$
\geq \sum_{d \in \mathcal{D}^{\mathsf{sat}}} \sum_{\substack{S: d \odot S \\ F \cap \delta(S) \neq \emptyset}} y_{S,d}
$$

$$
= \sum_{d \in \mathcal{D}^{\mathsf{sat}}} \sum_{S: d \odot S} y_{S,d},
$$

because $y_{S,d} = 0$ if $d \in \mathcal{D}^{\mathsf{sat}}$ and $F \cap \delta(S) = \emptyset$,

$$
= \sum_{d \in \mathcal{D}^{\mathsf{sat}}} y_d
$$

Proof. [Proof of Theorem 3.2] We know that $\mathsf{length}(F^+) + \pi(\mathcal{D} \setminus \mathcal{D}^+) \leq 3\mathsf{OPT}$ because we start with a 3-approximate solution. For any demand $d = (s, t)$, we know that y_d is not more than the distance of s, t in G^*. Since the distance between endpoints of d is zero if it is satisfied in $\mathcal{D}^+$, y_d is non-zero only if

$d \in \mathcal{D} \setminus \mathcal{D}^+$, we have $y(\mathcal{D}) = y(\mathcal{D} \setminus \mathcal{D}^+) \leq \pi^*(\mathcal{D} \setminus \mathcal{D}^+)$ by constraint (3.1). Lemma 3.1 gives $\mathsf{length}(F)$ in G^*, denoted by $\mathsf{length}_{G^*}(F)$, is at most $2y(\mathcal{D}) \leq 2\pi^*(\mathcal{D} \setminus \mathcal{D}^+) = 2\epsilon^{-1}\pi(\mathcal{D} \setminus \mathcal{D}^+) \leq 6\epsilon^{-1}\mathsf{OPT}$. Therefore $\mathsf{length}(F) = \mathsf{length}(F^+) + \mathsf{length}_{G^*}(F) \leq (6\epsilon^{-1} + 3)\mathsf{OPT}$.

To establish the second condition of the theorem, take an optimal forest F': F' satisfies demands $\mathcal{D}^{\mathsf{OPT}}$, and we have $\mathsf{length}(F') + \pi(\mathcal{D} \setminus \mathcal{D}^{\mathsf{OPT}}) = \mathsf{OPT}$. Define $A := \mathcal{D}^{\mathsf{OPT}} \setminus \mathcal{D}^{\mathsf{sat}}$ and $B := \mathcal{D}^{\mathsf{unsat}} \setminus A$. The penalty of F' under π' is $\pi((\mathcal{D} \setminus \mathcal{D}^{\mathsf{OPT}}) \cup \mathcal{D}^{\mathsf{unsat}}) = \pi((\mathcal{D}^{\mathsf{sat}} \setminus \mathcal{D}^{\mathsf{OPT}}) \cup A \cup B)$. Hence the increase in penalty of F' due to changing from π to π' is $\pi((\mathcal{D}^{\mathsf{sat}} \setminus \mathcal{D}^{\mathsf{OPT}}) \cup A \cup B) - \pi((\mathcal{D}^{\mathsf{sat}} \setminus \mathcal{D}^{\mathsf{OPT}}) \cup B) \leq \pi(A \cup B) - \pi(B)$ due to the decreasing marginal cost property of submodular functions. We have $y(A \cup B) = \pi^*(A \cup B) = \epsilon^{-1}\pi(A \cup B)$ because $A \cup B = \mathcal{D}^{\mathsf{unsat}}$ is the set of dead demands of `Submodular-PC-Clustering`; see the first condition of Lemma 3.1. We also have $\epsilon^{-1}\pi(B) = \pi^*(B) \geq y(B)$ because of Constraint (3.1). Therefore the additional penalty is at most $\epsilon[y(A \cup B) - y(B)] = \epsilon y(A)$. Since F' satisfies the demands A, we have $y(A) \leq \mathsf{length}(F') \leq \mathsf{OPT}$ from Lemma 3.3. Therefore, the additional penalty is at most $\epsilon\mathsf{OPT}$.

The extension to SPCTSP and SPCS is straightforward once we observe that the cost of building a tour or a stroll[9] on a subset of vertices is at least the cost of constructing a Steiner tree on the same set. Hence, there algorithm pretends it has an SPCST instance, and restricts the demand set accordingly. However, the extra penalty due to the ignored demands $\mathcal{D}^{\mathsf{unsat}}$ is charged to the Steiner tree cost which is no more than the TSP or stroll length.

3.5 Restricting the connectivity We first run `Restrict-Demands` on $(G, \mathcal{D}, \pi)$. Let F and $\mathcal{D}^{\mathsf{unsat}}$ be its output. The forest F satisfies all the demands in $\mathcal{D}^{\mathsf{sat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{unsat}}$. The length of this forest is $O(\mathsf{OPT})$ and the demands in $\mathcal{D}^{\mathsf{unsat}}$ can be safely ignored.

The forest F consists of tree components T_i. In the following, we connect some of these components to make the trees $\hat{T}_i$. It is easy to see that this construction guarantees the first two conditions of Theorem 3.3. We work on a graph $G^*(V^*, E^*)$ formed from G by contracting each tree component of F. A potential ϕ_v is associated with each vertex v of G^*, which is ϵ^{-1} times the length of the tree component corresponding to v in case v is the contraction of a tree component, and zero otherwise.

We use the algorithm `PC-Clustering` introduced in [13] to cluster the components T_i and construct a forest

---
[9]A *stroll* is similar to a tour, except that it may start and end on different vertices.

F_2 with components $\hat{T}_i$; the details of the algorithm can be seen in [13]. We obtain the folowing guarantees. Appendix D explains `PC-Clustering` for the sake of completeness.

LEMMA 3.4. ([13, LEMMA 6]) *The cost of F_2 is at most $2\sum_{v\in V^*}\phi_v$.*

Recall that the trees T_i are contracted in F_2. Construct $\hat{F}$ from F_2 by uncontracting all these trees. Let $\hat{F}$ consist of tree components $\hat{T}_i$. It is not difficult to verify that $\hat{F}$ is indeed a forest, but we do not need this condition since we can always remove cycles to find a forest. Define $\hat{\mathcal{D}}_i := \{(s,t)\in\mathcal{D}: s,t\in V(\hat{T}_i)\}$, and let $\mathcal{D}^*$ be the subset of demands satisfied by `OPT`. Define $\mathcal{D}_i^* := \mathcal{D}^*\cap\mathcal{D}_i$, and denote by $\mathsf{SteinerForest}(G,\mathcal{D})$ the length of a minimum Steiner forest of G satisfying the demands $\mathcal{D}$.

LEMMA 3.5. ([13, LEMMA 10])
$\sum_i \mathsf{SteinerForest}(G,\mathcal{D}_i^*) \leq (1+\epsilon)\mathsf{SteinerForest}(G,\mathcal{D}^*)$.

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

Proof. [Proof of Theorem 3.3] The first condition of the lemma follows directly from our construction: we start with a solution, and never disconnect one of the tree components in the process. The construction immediately implies the second condition. By Lemma 3.4, the cost of F_2 is at most $2\sum_{v\in V}\phi_v \leq \frac{2}{\epsilon}\mathsf{length}(F)$. Thus, $\hat{F}$ costs no more than $(2/\epsilon+1)\mathsf{length}(F)$, giving the third condition. Finally, Lemma 3.5 establishes the last condition.

4 PTASs for PCST, PCTSP and PCS on planar graphs

Since `PCST` is a special case of `PCSF`, Theorems 2.1 and 2.2 imply that `PCST` admits a PTAS on planar graphs. However, obtaining the same result for `PCTSP` and `PCS` is not immediate from those theorems since the latter problems are not special cases of `PCSF`. Here we explain how we can use these theorems to obtain the desired PTASs. Here we focus on `PCTSP`; however, the same arguments with minor changes apply to `PCS` as well.

Take an instance $\mathcal{I} = (G,\mathcal{D},\pi)$ of `PCTSP`, and apply Theorem 3.2 on $\mathcal{I}$ to obtain F and $\mathcal{D}^{\mathsf{unsat}}$. Since all the demands share a common root vertex[10], all the terminals in $\mathcal{D}^{\mathsf{sat}}$ are connected in F. We then invoke the TSP spanner construction of Arora et al. [6] to build H. Finally, we use the contraction decomposition

---

[10]If we have a penalty for each vertex in the `PCTSP` formulation, we can guess a root vertex r and define the demand pairs accordingly.

---

theorem of Demaine et al. [28] to contract a small-weight subset of edges and reduce the problem to graphs of bounded treewidth. The total additional charge due to penalties of $\mathcal{D}^{\mathsf{unsat}}$ and contracted edges is at most $O(\epsilon)\mathsf{OPT}$. Therefore we can obtain a PTAS by solving the bounded-treewidth instance precisely.

5 Hardness of PCSF on series-parallel graphs

We first present the hardness proof for `PCSF` on a planar graph of treewidth two. The proof shows hardness for a very restricted class of graphs: short cycles going through a single central vertex.

Proof. [Proof of Theorem 2.3(1)] We reduce an instance $\mathcal{I}$ of `Vertex Cover` on 3-regular graphs to an instance $\mathcal{I}'$ of `PCSF` on a planar graphs of treewidth two. The former is known to be APX-hard [3]. The instance $\mathcal{I}$ is defined by an undirected graph G. If n denotes the number of vertices of G, the number edges is $m = 3n/2$. We will denote the i-th vertex of G by v_i, the j-th edge by e_j, and the first and second endpoints of e_j by $e_j^{(1)}$ and $e_j^{(2)}$, respectively.

We now specify the reduction (illustrated in Figure 1); $\mathcal{I}'$ is represented by $(H,\mathcal{D},\pi)$. The graph H consists of the vertices

- a_i for $1 \leq i \leq n$,
- b_j, c_j^1, c_j^2 for $1 \leq j \leq m$,
- central vertex w,

and the edges

- $\{w,a_i\}$ of cost 2 ($1 \leq i \leq n$),
- $\{w,c_j^1\}, \{w,c_j^2\}, \{c_j^1,b_j\}, \{c_j^2,b_j\}$ of cost 1 ($1 \leq j \leq m$).

The instance contains the following demands:

- $\{w,b_j\}$ with penalty 3 ($1 \leq j \leq m$),
- If $v_i = e_j^{(\ell)}$ for some $1 \leq i \leq n$, $1 \leq j \leq m$, and $\ell \in \{1,2\}$, then $\{a_i,c_j^\ell\}$ is a demand with penalty 1.

Thus the number of demands is exactly $m + 3n$ and each a_i appears in exactly 3 demands. We claim that the cost of the optimum solution of $\mathcal{I}'$ is exactly $2m+2n+\tau(G)$, where $\tau(G)$ is the size of the minimum vertex cover in G. Note that $\tau(G) \geq n/3$ (as G is 3-regular), thus $2m+2n+\tau(G)$ is at most a constant times $\tau(G)$. In order to prove the correctness of the reduction, we prove the following two statements:
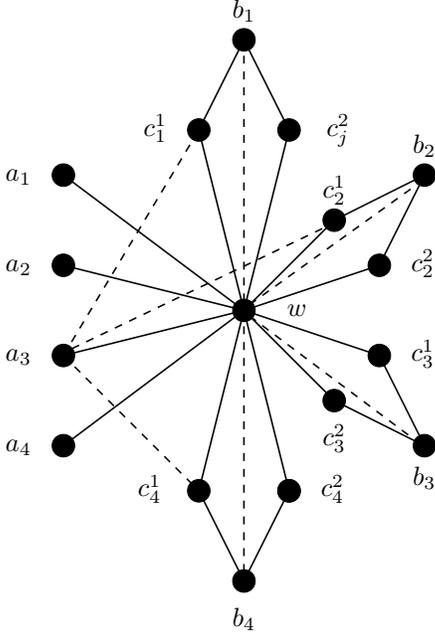
Figure 1: Illustrating the reduction from 3-Regular Vertex Cover to PCSF.

(1) Given a vertex cover of size k for G, a solution of cost $2m + 2n + k$ can be constructed.

(2) Given a solution of cost at most $2m + 2n + k$, a vertex cover of size at most k can be constructed.

To prove (1), suppose that C is a vertex cover of size k for G. Let T be a tree of H that contains

- edge $\{w, a_i\}$ if and only if $v_i \notin C$,
- edges $\{w, c_j^1\}, \{c_j^1, b_j\}$ if and only if $e_j^1 \notin C$,
- edges $\{w, c_j^2\}, \{c_j^2, b_j\}$ if and only if $e_j^1 \in C$.

The total cost of T is $2(n-k)+2m$. Observe that all the demands $\{w, b_j\}$ are connected (either via c_j^1 or c_j^2). Furthermore, if $v_i \notin C$, then all three demands where a_i appears are satisfied: edge $\{w, a_i\}$ is in T and if $v_i = e_j^1$, then edge $\{w, c_j^1\}$ is in T as well. (Note that if $v_i = e_j^2$ and $v_i \notin C$, then $e_j^1 \in C$ must hold, and therefore $\{w, c_j^2\}$ is in T.) Thus the total penalty is at most $3k$, and hence the cost of the solution is at most $2n + 2m + k$, as claimed.

To prove (2), suppose that subgraph F of G is a solution such that the sum of the cost of F and the penalties is at most $2m + 2n + k$. We can assume that for every $1 \leq i \leq n$, vertex b_j can be reached from w: otherwise we can decrease the penalty by 3 at the cost of adding two edges of cost 1. Furthermore, we can assume

that only one of c_j^1 and c_j^2 is can be reached from w: otherwise we can remove an edge without disconnecting b_j from w, thus the cost decreases by 1 and the penalty increases by at most 1. Finally, we can assume that if $\{w, a_i\} \in F$, then all 3 demands containing a_i are connected: otherwise removing $\{w, a_i\}$ decreases the cost by 2 and increases the penalty by at most 2.

Let vertex v_i be in C if and only if $\{w, a_i\} \notin F$. We claim that C is a vertex cover of size at most k. To see that C is a vertex cover, consider an edge e_j. We have observed above that one of c_j^1 and c_j^2 cannot be reached from w. If c_j^1 cannot be reached from w and $e_j^{(1)} = v_i$, then the demand $\{v_i, c_j^1\}$ is not connected by F. Therefore, not all 3 demands containing a_i are connected, which means (as observed above) that $\{w, a_i\} \notin F$. Thus $v_i \in C$, covering the edge e_j.

Since every b_j can be reached from w and $\{w, a_i\} \in F$ if $v_i \notin C$, the cost of F is at least $2m + 2(n - |C|)$. Furthermore, if $v_i \in C$, then $\{w, a_i\} \notin F$, which means that we have to pay the penalty for the 3 demands containing a_i. Therefore, the total cost of the solution is at least $2m + 2n + |C|$. We assumed that the cost of the solution is at most $2m + 2n + |C|$, thus $|C| \leq k$ follows, what we had to prove.

5.1 Hardness of Euclidean PCSF The proof for the Euclidean version is very similar to the graph version. The main difference is that the central vertex w is replaced by a set of points arranged along a long vertical path.

Proof. [Proof of Theorem 2.3(2)] We reduce an instance $\mathcal{I}$ of Vertex Cover on 3-regular graphs to an instance $\mathcal{I}'$ of PCSF on points in the Euclidean plane. If n denotes the number of vertices of the 3-regular graph G in $\mathcal{I}$, then the number edges is $m = 3n/2$. We will denote the i-th vertex of G by v_i, the j-th edge by e_j, and the first and second endpoints of e_j by $e_j^{(1)}$ and $e_j^{(2)}$, respectively.

We now specify the reduction (illustrated in Figure 2). Let us define $U := 10000(n + m)$ ("basic unit of cost"), $H = 10U$ ("horizontal length"), and $V = 100U$ ("vertical spacing"). Instance $\mathcal{I}'$ contains the following set P of points:

- $z_{0,y} = (0, y)$ for every $-mV \leq y \leq nV$,
- $z_{x,y} = (x, y)$ and for every $0 \leq x \leq H$ and $y = iV$ for $1 \leq i \leq n$,
- $z_{x,y} = (x, y)$ and $z_{x,y+4U}$ for every $0 \leq x \leq H$ and $y = -jV$ for $1 \leq j \leq m$,
- $a_i = (H + 2U, iV)$ for $1 \leq i \leq n$,

- $b_j = (H, -jV + 2U)$ for $1 \le j \le m$,

- $c_j^1 = (H, -jV + U)$, and $c_j^2 = (H, -jV + 3U)$ for $1 \le j \le m$.

Let Z be the set of all $z_{x,y}$ vertices in P, note that $|Z| = V(i + j) + 1 + (i + 2j)H$. For ease of notation, we define $w_i = z_{H,iV}$, $w_j^1 = z_{H,-jV}$, $w_j^2 = z_{H,-jV+4U}$.

The instance contains the following demands:

1. If $z_{x,y}$ and $z_{x+1,y}$ are both in P, then there is a demand $\{z_{x,y}, z_{x+1,y}\}$ with penalty 1.

2. If $z_{x,y}$ and $z_{x,y+1}$ are both in P, then there is a demand $\{z_{x,y}, z_{x,y+1}\}$ with penalty 1.

3. $\{(0,0), b_j\}$ with penalty $3U$ $(1 \le j \le n)$,

4. If $v_i = e_j^{(\ell)}$ for some $1 \le i \le n$, $1 \le j \le m$, and $\ell \in \{1, 2\}$, then $\{a_i, c_j^\ell\}$ is a demand with penalty $U - 10$.

The total number of demands is $|Z| - 1 + n + 3m$ and each a_i appears in exactly 3 demands. We claim that the cost of the optimum solution of $\mathcal{I}'$ is between $|Z| + (2m + 2n + \tau(G))U$ and $|Z| + (2m + 2n + \tau(G))U - 100n$, where $\tau(G)$ is the size of the minimum vertex cover in G. Note that $m = 3n/2$ and $\tau(G) \ge m/3$, thus $|Z| + (2m + 2n + \tau(G))U$ is at most a constant factor larger than $\tau(G)U$.

More precisely, in order to prove the correctness of the reduction, we prove the following two statements:

(1) Given a vertex cover of size k for G, a solution of cost at most $|Z| + (2m + 2n + k)U$ for $\mathcal{I}'$ can be constructed.

(2) Given a solution of cost at most $|Z| + (2m + 2n + k)U$ for $\mathcal{I}'$, a vertex cover of size at most k can be constructed.

To prove (1), suppose that C is a vertex cover of size k for G. Let F be the forest (actually, a tree) that contains

1. edge $\{z_{x,y}, z_{x+1,y}\}$ if both these points are in P,

2. edge $\{z_{x,y}, z_{x,y+1}\}$ if both these points are in P,

3. edge $\{w_i, a_i\}$ if $v_i \notin C$,

4. edges $\{w_j^1, c_j^1\}$ and $\{c_j^1, b_j\}$ if $e_j^{(1)} \notin C$,

5. edges $\{w_j^2, c_j^2\}$ and $\{c_j^2, b_j\}$ if $e_j^{(1)} \in C$.

The total cost of F is $|Z| - 1 + 2U(n - k) + 2Um$. Observe that all the demands $\{(0,0), b_j\}$ are satisfied. Furthermore, if $v_i \notin C$, then all three demands where
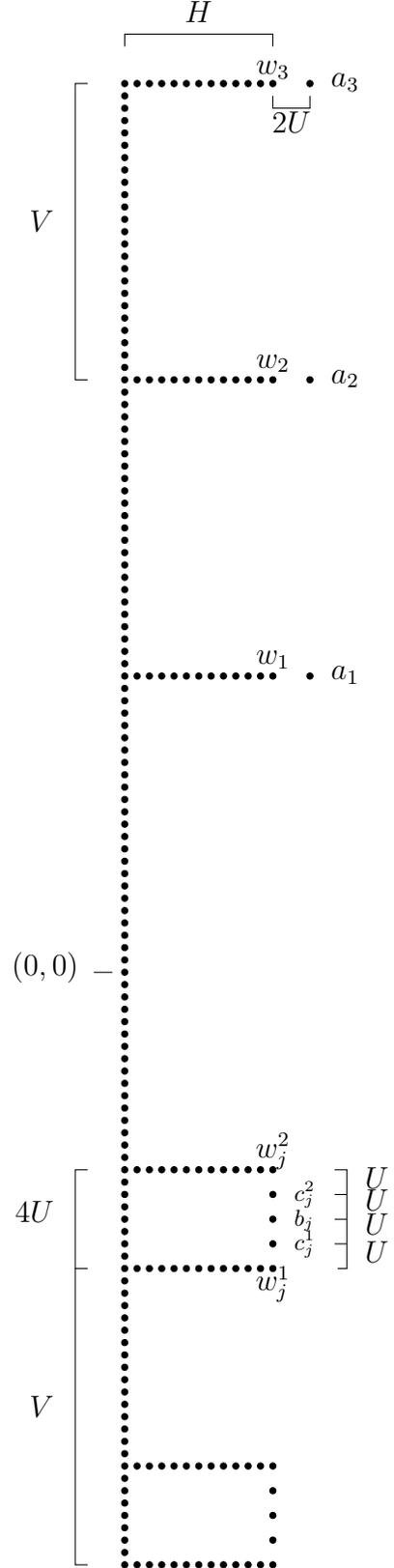
Figure 2: Illustrating the reduction from 3-Regular Vertex Cover to Euclidean PCSF.

a_i appears are satisfied. This can be seen as follows. First, a_i is in the same component as w_i and hence as every vertex of Z. If $v_i = e_j^{(1)}$, then there is a demand $\{a_i, c_j^1\}$ and c_j^1 is connected with w_j^1 (and hence with a_i). If $v_i = e_j^{(2)}$, then $v_i \notin C$ means that $e_j^{(1)} \in C$ must hold, and therefore c_j^2 is connected to w_j^2, satisfying the demand $\{a_i, c_j^2\}$. Thus the total penalty is at most $3k(U - 10)$, and hence the cost of the solution is at most $|Z| - 1 + (2m + 2n + k)U - 30k$, as claimed.

To prove (2), suppose that forest F is an optimum solution such that the sum of the cost of F and the penalties is at most $|Z| + (2n + 2m + k)U$. First, we can assume that every demand of the first two types is satisfied: if, say, $(z_{x,y}, z_{x+1,y})$ is not satisfied, then we can extend F by adding an edge of cost 1, which decreases the penalty by at least 1. Thus all the $z_{x,y}$ points are in the same connected component K of F. We can also assume that every demand of the third type is satisfied: if $\{(0,0), b_j\}$ is not satisfied, then we can decrease the penalty by $3U$ at the cost of $2U$ by adding edges $\{w_j^1, c_j^1\}$ and $\{c_j^1, b_j\}$, contradicting the optimality of F. Therefore, every vertex b_j is in the component K.

Let $Z' = \{z_{x,y} \in Z \mid x = 0 \vee x \geq 10\}$. Let R be the region of the plane at Manhatten distance at most 3 from Z'. Note that R consists of one "vertical" and $n + 2m$ "horizontal" components.

We claim that the cost of F inside R is at least $|Z'|$. We have seen above that a single component K of F contains every point of $P \cap R$. The restriction of K to R gives rise to several components. Consider such a component K' containing a subset $S \subseteq Z'$ of vertices. We show that the cost of K' is at least $|S|$. The vertices of S lie on a horizontal or vertical line. This means that there are two vertices $s_1, s_2 \in S$ at distance $d \geq |S| - 1$. As K is not contained fully in any component of R, component K' has to contain a point s_3 on the boundary of R. As s_3 is at distance at least 3 from s_1 and s_2, it can be verified that any Steiner tree of s_1, s_2, s_3 has cost at least $d + 1 = |S|$. Summing for every component K' of the restriction of K to R, we get that the cost of K in R is at least $|P \cap R|$.

Let R^+ be the region of space at Manhattan distance at most 3 from Z. We claim that the cost of every component of $F \setminus R^+$ is at most $3U$. There are two types of components of $F \setminus R^+$: (1) those that contain a point of P and (2) those that do not contain such a point. Clearly, there are at most $n + 3m$ components of the first type. Suppose that there is a component D of the second type having cost more than $3U$. In this case, we modify F to obtain a better solution as follows. Consider $F \setminus R^+$ (i.e., let us remove the part of F inside R^+) and let us remove every component of the second

type. After that, let us add all the $|Z| - 1$ edges of the form $\{w_{x,y}, w_{x+1,y}\}$, $\{w_{x,y}, w_{x,y+1}\}$. Finally, for every component of the first type, if it intersects R^+, then let us choose a point of the component on the boundary of R^+ and connect this point to the nearest vertex of Z. It is clear that the new forest F' satisfies every demand satisfied by F: every point of P connected to Z remains connected to Z. By our claim in the previous paragraph, the cost of $F \setminus R'$ is less than the cost of F by at least $|Z'| = |Z| - 9(n + 2m)$. Removing components of the second type decreases the cost by more than $3U$ (as there are at least one such component having cost more than $3U$). The edges connecting Z increase the cost by $|Z| - 1$. Adding the new connections corresponding to the components of the first type increases the cost by at most $n + 3m$. As $3U \geq 9(n + 2m) - 1 + n + 3m$, forest F' is a strictly better solution, a contradiction.

Suppose now that there is a component D of the first type with cost more than $3U$. For $-m \leq s \leq n$, let R_s be the region of the plane at Manhattan distance at most $4U$ from (H, sV). Observe that for each s, all the points of $P \cap R_s$ can be connected to the nearest point of Z with a total cost of at most $3U$. This means that if D intersects only one of these regions, say R_s, then we can substitute D at cost at most $3U$ in such a way that every demand satisfied by F remains satisfied, contradicting the optimality of F. Suppose therefore that D intersects $t \geq 2$ of these regions; in this case, the cost of D is at least $(t-1)(V - 8U) > 6tU - 6U \geq 3tU$. Let us replace D by connecting every point of $P \cap D$ to the closest vertex of Z. The new connections increase the cost by at most $t \cdot 3U$, which is less than the cost of D, a contradiction.

We have proved that for every component D of $F \setminus R^+$, $D \cap P$ is either a single a_i, or a subset of $\{b_j, c_j^1, c_j^2\}$. Therefore, every such component D intersects R^+: otherwise, D could be safely removed, as it does not satisfy any demand. Next we show that it can be asssumed that only one of c_j^1 and c_j^2 is in K. Otherwise we can remove every component of $F \setminus R^+$ intersecting $\{b_j, c_j^1, c_j^2\}$ and replace them with the edges $\{w_j^1, c_j^1\}$ and $\{c_j^1, b_j\}$. The total cost of the components we removed is at least $2U - 3 + U - 3$ (which is the minimum cost of connecting b_j, c_j^1, c_j^2 to R^+) and the new edges have cost $2U$. This transformation might disconnect the demand containing c_j^2, hence the penalty can increase by at most $U - 10$ only, contradicting the optimality of F.

We can assume that if a_i is in K, then all 3 demands containing a_i are connected: otherwise removing the component of $F \setminus R^+$ containing a_i decreases the cost by at least $2U - 3$ and increases the penalty by at most $2(U - 10)$.

Let vertex v_i be in C if and only if a_i is not in component K. We claim that C is a vertex cover of size at most k. To see that C is a vertex cover, consider an edge e_j. We have observed above that one of c_j^1 and c_j^2 is not in K. If $c_j^1 \notin K$ and $e_j^{(1)} = v_i$, then the demand $\{a_i, c_j^1\}$ is not connected by F. Therefore, not all 3 demands containing a_i are connected, which means (as observed above) that a_i is not in K. Thus $v_i \in C$, covering the edge e_j. Similarly, $c_j^2 \notin K$, then $e_j^{(2)} \in C$.

The cost of $F \cap R^+$ is at least $|Z| - 9(n + 2m)$. Since every b_j is in K and a_i is in K if $v_i \notin C$, the cost of $F \setminus R^+$ is at least $(2U - 3)m + (2U - 3)(n - |C|)$. Furthermore, if $v_i \in C$, then we have to pay the penalty for the 3 demands containing a_i. Therefore, the total cost of the solution is at least

$$|Z| - 9(n+2m) + (2U-3)m + (2U-3)(n-|C|) + 3|C|(U-10)$$

$$\geq |Z| + (2m + 2n + |C|)U - 100n$$

We assumed that the cost of the solution is at most $|Z| + (2m + 2n + k)U$. As $U > 100n$, this is only possible if $|C| \leq k$, what we had to prove.

References

[1] A. Agrawal, P. N. Klein, and R. Ravi, *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), 1991, pp. 134–144.

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

[3] P. Alimonti and V. Kann, *Some APX-completeness results for cubic graphs*, Theoret. Comput. Sci., 237 (2000), pp. 123–134.

[4] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff, *Improved approximation algorithms for prize-collecting Steiner tree and TSP*, SIAM Journal on Computing, (to appear). Preliminary version in *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 427–436, 2009.

[5] A. Archer, A. Levin, and D. P. Williamson, *A faster, better approximation algorithm for the minimum latency problem*, SIAM J. Comput., 37 (2008), pp. 1472–1498.

[6] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn, *A polynomial-time approximation scheme for weighted planar graph tsp*, in Proceedings of the ninth annual ACM-SIAM Symposium on Discrete algorithms (SODA'98), 1998, pp. 33–41.

[7] S. Arora and G. Karakostas, *A $2 + \epsilon$ approximation algorithm for the k-MST problem*, Mathematical Programming, 107 (2006), pp. 491–504.

[8] S. Arya and H. Ramesh, *A 2.5 factor approximation algorithm for the k-MST problem*, Information Processing Letters, 65 (1998), pp. 117–118.

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

[10] E. Balas, *The prize collecting traveling salesman problem*, Networks, 19 (1989), pp. 621–636.

[11] N. Bansal, A. Blum., S. Chawla, and A. Meyerson, *Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), ACM New York, NY, USA, 2004, pp. 166–174.

[12] M. Bateni and M. Hajiaghayi, *Euclidean prize-collecting steiner forest*, in Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN'10), 2010. to appear.

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

[14] M. Bateni, M. Hajiaghayi, and D. Marx, *Prize-collecting network design on planar graphs*, CoRR, abs/1006.4339 (2010).

[15] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. P. Williamson, *A note on the prize collecting traveling salesman problem.*, Mathematical Programming, 59 (1993), pp. 413–420.

[16] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff, *Approximation Algorithms for Orienteering and Discounted-Reward TSP*, SIAM Journal on Computing, 37 (2007), pp. 653–670. Preliminary version in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 46–55, 2003.

[17] H. L. Bodlaender and A. M. C. A. Koster, *Combinatorial optimization on graphs of bounded treewidth*, Comput. J., 51 (2008), pp. 255–269.

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

[19] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanita, *An improved LP-based approximation for steiner tree*, in Proceedings of the fourty-second annual ACM Symposium on Theory of computing (STOC'10), New York, NY, USA, 2010, ACM. to appear.

[20] S. Cabello and B. Mohar, *Finding shortest non-separating and non-contractible cycles for topologically embedded graphs*, in Proceedings of the 13th Annual European Symposium of Algorithms (ESA), 2005,

pp. 131–142.

[21] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, *Paths, Trees, and Minimum Latency Tours*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2003, pp. 36–45.

[22] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, *Paths, trees, and minimum latency tours*, in Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science, 2003, pp. 36–45.

[23] C. Chekuri, A. Ene, and N. Korula, *Prize-collecting steiner tree and forest in planar graphs*, CoRR, abs/1006.4357 (2010).

[24] C. Chekuri, N. Korula, and M. Pál, *Improved Algorithms for Orienteering and Related Problems*, ACM Transactions on Algorithms, (to appear). Preliminary version in *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 661–670, 2008.

[25] N. Christofides, *Worst-case analysis of a new heuristic for the travelling-salesman problem*, tech. rep., Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[26] F. A. Chudak, T. Roughgarden, and D. P. Williamson, *Approximate k-MSTs and k-Steiner trees via the primal-dual method and lagrangean relaxation*, in Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO'01), London, UK, 2001, Springer-Verlag, pp. 60–70.

[27] F. A. Chudak, T. Roughgarden, and D. P. Williamson, *Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation*, Mathematical Programming, 100 (2004), pp. 411–421.

[28] E. D. Demaine, M. Hajiaghayi, and B. Mohar, *Approximation algorithms via contraction decomposition*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 278–287.

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

[30] N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, 1996, pp. 302–309.

[31] ——, *Saving an epsilon: a 2-approximation for the k-MST problem in graphs*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 396–402.

[32] M. Goemans, *The prize-collecting TSP revisited*. Available from http://www-math.mit.edu/ goemans/prizecollect.ps. Talk slides from the 1998 SIAM Discrete Math conference.

[33] M. X. Goemans, *Combining approximation algorithms for prize-collecting TSP*. unpublished manuscript, 2009.

[34] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM Journal on Computing, 24 (1995), pp. 296–317.

[35] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. Assoc. Comput. Mach., 42 (1995), pp. 1115–1145.

[36] M. Grigni, E. Koutsoupias, and C. Papadimitriou, *An approximation scheme for planar graph tsp*, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95), Washington, DC, USA, 1995, IEEE Computer Society, p. 640.

[37] M. Hajiaghayi and K. Jain, *The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, New York, 2006, ACM, pp. 631–640.

[38] M. Hajiaghayi, R. Khandekar, G. Kortsarz, and Z. Nutov, *Prize-collecting Steiner network problems*, in Proceedings of the 14th Conference on Integer Programming and Combinatorial Optimization (IPCO), 2010. to appear.

[39] J. Hoogeveen, *Analysis of Christofides' heuristic: Some paths are more difficult than cycles*, Operations Research Letters, 10 (1991), pp. 291–295.

[40] S. Iwata, L. Fleischer, and S. Fujishige, *A combinatorial strongly polynomial algorithm for minimizing submodular functions*, J. ACM, 48 (2001), pp. 761–777 (electronic).

[41] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[42] D. S. Johnson, M. Minkoff, and S. Phillips, *The prize collecting Steiner tree problem: theory and practice.*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 760–769.

[43] P. N. Klein, *A subset spanner for planar graphs, with application to subset TSP*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), 2006, pp. 749–756.

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

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

[46] V. Nagarajan and R. Ravi, *Poly-logarithmic Approximation Algorithms for Directed Vehicle Routing Problems*, in Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), 2007, pp. 257–270.

[47] N. Robertson and P. D. Seymour, *Graph minors. II. algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309–322.

[48] ——, *Graph minors. XI. circuits on a surface*, Journal

of Combinatorial Theory, Series B, 60 (1994), pp. 72–106.

[49] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian, *Approximating the single-sink link-installation problem in network design*, SIAM J. on Optimization, 11 (2000), pp. 595–610.

[50] Y. Sharma, C. Swamy, and D. P. Williamson, *Approximation algorithms for prize collecting forest problems with submodular penalty functions*, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '07), 2007, pp. 1275–1284.

A Basic graph theory 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 $\mathsf{dist}_G(x, y)$. For an edge set E, we denote by $\ell(E) := \sum_{e \in E} c_e$ the total length of edges in E.

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

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 [47]. 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) for every $v \in V$, the set of nodes $\{i \in I \mid v \in B_i\}$ forms a connected subtree of T.

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 $\mathsf{tw}(G)$, 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 also need a basic notion of embedding; see, e.g., [48, 20]. 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 [45]. 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.

B PCST, PCTSP and PCS on bounded-treewidth graphs

Treewidth is a notion of how similar a graph is to trees. Since tree structure usually lends itself to the dynamic programming approach, it is plausible that many optimization problems may be solvable in polynomial time on graphs of bounded treewidth; Bodlaender and Koster [17] have a comprehensive survey on

this topic. In particular, several Steiner network problems become relatively easy when restricted to bounded-treewidth graphs. Among them are Steiner Tree, TSP and Stroll. One surprising outlier is Steiner forest that is proved to be NP-hard, yet it admits a PTAS [13]. In this section, we study the prize-collecting extensions of the above problems, and when possible, we provide a polynomial-time algorithm for them. More specifically, we present PTASs for PCST, PCTSP and PCS on bounded-treewidth graphs. We already showed in Section 5 that PCSF is APX-hard even on series-parallel graphs. The proof is extended to give APX-hardness for Euclidean plane.

We focus the discussion on PCST, however, minor modifications allow us to solve PCTSP and PCS, too. We are given a weighted graph $G(V, E)$ of treewidth $k - 1$ for a fixed parameter k, and a penalty function $\pi : V \to \mathbb{R}_+$. We have a nice tree decomposition $(T, \mathcal{B})$ for G. Each bag B_i has size at most k. These are sometimes called *portals* for the subtree below node B_i. Let I denote the nodes of the tree decomposition T, and for each $i \in I$, let T_i be the subtree of T below i. A dynamic programming entry is specified by a tuple $(i, S, \mathcal{P})$ where

- $i \in I$ is a node in the tree decomposition,

- $S \subseteq B_i$ is a subset of portals of the subtree T_i, and

- $\mathcal{P}$ is a partition of S.

Let us denote by V_i the vertices corresponding to the subtree T_i, i.e., $V_i := \cup_{i' \in T_i} B_{i'}$. A dynamic programming entry $\mathrm{DP}(i, S, \mathcal{P})$ takes up the least cost of building a subgraph H such that

- H uses only the edges whose both endpoints are in V_i,

- H connects the vertices in each set P_j of the partition $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$,

- S is the subset of B_i whose penalty is not paid, moreover, if a vertex $v \in V_i$ is not connected to S via H, then its penalty $\pi(v)$ is paid in the total cost.

The final solution to the problem can be found as $\min_S \mathrm{DP}(r, S, \{S\})$ where r is the root of the tree decomposition, i.e., it does not matter which subset of the bag of the root is picked as long as they form a single component.

The DP entries are easy to compute for leaves: let $B_i = \{v\}$ for a leaf i. There are two possibilities: $\mathrm{DP}(i, \emptyset, \emptyset) = \pi(v)$ and $\mathrm{DP}(i, \{v\}, \{\{v\}\}) = 0$. The update procedure works as follows for different tree nodes:

Introduce node i is the parent of i', and we have $B_i = B_{i'} \cup \{v\}$. Then, $\mathrm{DP}(i, S, \mathcal{P}) = \pi(v) + \mathrm{DP}(i', S, \mathcal{P})$ if $v \notin S$. Next consider an entry $\mathrm{DP}(i, S, \mathcal{P})$ such that for $v \in S$ and $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ where $v \in P_1$. Let $\mathcal{P}' := \{P_1 \setminus \{v\}, P_2, \ldots, P_m\}$ and let d be the distance of v to the set $P_1 \setminus \{v\}$. The dynamic programming sets $\mathrm{DP}(i, S, \mathcal{P}) = d + \mathrm{DP}(i', S \setminus \{v\}, \mathcal{P}')$.

Forget node i is the parent of i', and we have $B_{i'} = B_i \cup \{v\}$. Then,

$$
\mathrm{DP}(i, S, \mathcal{P}) = \min \begin{cases} \pi(v) + \mathrm{DP}(i', S, \mathcal{P}) \\ \min_{\mathcal{P}'} \{\mathrm{DP}(i', S \cup \{v\}, \mathcal{P}') : \mathcal{P}' \\ \quad \text{is formed by adding } v \text{ to a} \\ \quad \text{set of } \mathcal{P}\} \end{cases}
$$

The first terms considers the case where we pay the penalty for v and do not connect it in the final Steiner tree, whereas the second term takes into account the case where v is connected to each connected component of the partition.

Join node the node i has two children i_1 and i_2 with the same bags. We set $\mathrm{DP}(i, S, \mathcal{P})$ to

$$
\min_{\mathcal{P}_1, \mathcal{P}_2} \{\mathrm{DP}(i_1, S, \mathcal{P}) + \mathrm{DP}(i_2, S, \mathcal{P}) - \pi(B_i \setminus S)\},
$$

where the minimization goes over all pairs $\mathcal{P}_1$ and $\mathcal{P}_2$ whose connectivity implies that of $\mathcal{P}$. The last term in the minimum operand is for canceling the double charging of the unsatisfied terminals of B_i.

It is not difficult to verify that the algorithm produces the correct output, and we defer the proof to the full version of the paper. The running time of the algorithm is polynomial in the number of DP entries, and the latter is at most $n \cdot 2^k \cdot k^k$. Since k is a constant, the running time is a polynomial.

To extend the algorithm to PCTSP, the DP state is modified to $(i, \mathcal{P})$ where $i \in I$ is a node of the tree decomposition, and $\mathcal{P}$ is a set of pairs of vertices in bag B_i. A pair s, t implies that there is a path between s and t in the subsolution, but the two nodes should be extended from outside the subtree T_i to make a tour. The final solution is stored in $\mathrm{DP}(r, \{(r, r)\})$. The algorithm for PCS works in the same way except that the final solution can be founded in $\min_{s,t \in B_r} \mathrm{DP}(r, \{(s, t)\})$ since we do not need to have a closed tour.

C   Missing proofs from Section 3

Proof. [Proof of Lemma 3.2] 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 jth event point occurs in the growth phase $(0 = t_0 \leq t_1 \leq t_2 \leq \cdots)$, so the jth 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 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 (3.1) became tight, we can exactly apportion the cost c_e amongst the collection of clusters $\{C : e \in \delta(C)\}$ whose variables "pay for" the edge, and can divide this up further by epoch. In other words, $c_e = \sum_j \sum_{C:e\in\delta(C)} y_C^{(j)}$. We will now prove that the total edge cost from F_2 that is apportioned to epoch j is at most $2\sum_C y_C^{(j)}$. In other words, during each epoch, the total rate at which edges of F_2 are paid for by all active clusters is at most twice the number of active clusters. Summing over the epochs yields the desired conclusion.

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

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 live clusters 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.

Proof. [Proof of Lemma 3.3] The length of the graph F is

$$\sum_{e\in F} c_e \geq \sum_{e\in F}\sum_{S:e\in\delta(S)} y_S \qquad \text{by (3.1)}$$
$$= \sum_S |F\cap\delta(S)|y_S$$
$$\geq \sum_{S:F\cap\delta(S)\neq\emptyset} y_S$$
$$= \sum_{S:F\cap\delta(S)\neq\emptyset}\sum_{d:d\odot S} y_{S,d}$$
$$= \sum_d \sum_{\substack{S:d\odot S \\ F\cap\delta(S)\neq\emptyset}} y_{S,d}$$
$$\geq \sum_{d\in\mathcal{D}^{\mathsf{sat}}} \sum_{\substack{S:d\odot S \\ F\cap\delta(S)\neq\emptyset}} y_{S,d}$$
$$= \sum_{d\in\mathcal{D}^{\mathsf{sat}}}\sum_{S:d\odot S} y_{S,d},$$

because $y_{S,d} = 0$ if $d \in \mathcal{D}^{\mathsf{sat}}$ and $F \cap \delta(S) = \emptyset$,

$$= \sum_{d\in\mathcal{D}^{\mathsf{sat}}} y_d$$

D Overview of `PC-Clustering`

The following LP has a variable $y_{S,v}$ for each $v \in S \subseteq V^*$.

$$\sum_{S:e\in\delta(S)}\sum_{v\in S} y_{S,v} \leq c_e \qquad \forall e \in E^*,$$
$$\sum_{S\ni v} y_{S,v} \leq \phi_v \qquad \forall v \in V^*,$$
$$y_{S,v} \geq 0 \qquad \forall v \in S \subseteq V^*.$$

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

The solution is built up in two stages. First we perform an *unrooted growth* to find a forest F_1 and a corresponding y vector. In the second stage, we *prune* some of the edges of F_1 to get another forest F_2. Uncontracting the trees T_i turns F_2 into the Steiner trees $\hat{T}_i$ in the statement of Theorem 3.3. The details of the two phases can be seen in Algorithm 3 (`PC-Clustering`). The proofs of some of the lemmas in this section are similar to the discussion in Section 3 or to the previous work [13].

Algorithm 3 PC-Clustering

Input: An instance $(G, \mathcal{D}, \pi)$ of Generalized prize-collecting Steiner forest
Output: Set of trees $\hat{T}_i$ with associated demands $\mathcal{D}_i$ and $\mathcal{D}'$.

1: Run Restrict-Demands on $(G, \mathcal{D}, \pi)$ to obtain F and $\mathcal{D}'$.
2: Let F consist of tree components $T_1, \ldots, T_k$.
3: Contract each tree T_i to build a new graph $G^*(V^*, E^*)$.
4: For any $v \in V^*$, let ϕ_v be ϵ^{-1} times the length of the tree T_i corresponding to v, and zero if there is no such tree.
5: Let $F_1 \leftarrow \emptyset$.
6: Let $y_{S,v} \leftarrow 0$ for any $v \in S \subseteq V^*$.
7: Let $\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V^*\}$.
8: **while** there is a live vertex **do**
9:     Let η be the largest possible value such that simultaneously increasing y_C by η for all active clusters C does not violate Constraints (D.1)-(D.1).
10:     Let $y_{C,v} \leftarrow y_{C,v} + \frac{\eta}{\kappa(C)}$ for all live vertices v in an active cluster C.
11:     **if** $\exists e \in E^*$ that is tight and connects two clusters **then**
12:         Pick one such edge $e = (u, v)$ connecting two clusters C_1 and C_2.
13:         Let $F_1 \leftarrow F_1 \cup \{e\}$.
14:         Let $C \leftarrow C_1 \cup C_2$.
15:         Let $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{C_1, C_2\}$.
16:         Let $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$.
17: Let $F_2 \leftarrow F_1$.
18: Let $\mathcal{B}$ be the set of all clusters $S \in \mathcal{S}$ such that $\sum_{v \in S} y_{S,v} = \sum_{v \in S} \phi_v$.
19: **while** $\exists S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$ for an edge e **do**
20:     Let $F_2 \leftarrow F_2 \setminus \{e\}$.
21: Construct F from F_2 by uncontracting all the trees T_i.
22: Let $\hat{F}$ consist of tree components $\hat{T}_i$.
23: Output $\mathcal{D}'$ and the set of trees $\{\hat{T}_i\}$, along with $\hat{\mathcal{D}}_i := \{(s,t) \in \mathcal{D} \setminus \mathcal{D}' : s, t \in V(\hat{T}_i)\}$.

We first bound the cost of the forest F_2. The following lemma is similar to the analysis of the algorithm in [35]. However, we do not have a primal LP to give a bound on the dual. Rather, the upper bound for the cost is the sum of all the potential values $\sum_v \phi_v$. In addition, we bound the cost of a forest F_2 that may have more than one connected component, whereas the prize-collecting Steiner tree algorithm of [35] finds a connected graph at the end.

The following lemma gives a sufficient condition for two vertices that end up in the same component of F_2.

LEMMA D.1. *Two vertices u and v of V^* are connected via F_2 if there exist sets S, S' both containing u, v such that $y_{S,v} > 0$ and $y_{S',u} > 0$.*

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

Consider a pair (v, S) with $y_{S,v} > 0$. If subgraph G' of G^* has an edge that goes through the cut $(S, \bar{S})$, at least a portion of length $y_{S,v}$ of G' is colored with the color v due to the set S. Thus, if G' cuts all the sets S for which $y_{S,v} > 0$, we can charge part of the length of G' to the potential of v. Later in Lemma 3.5, we are going to use potentials as a lower bound on the optimal solution. More formally, we say a graph $G'(V^*, E')$ *exhausts a color* u if and only if $E' \cap \delta(S) \neq \emptyset$ for any $S : y_{S,u} > 0$. The proof of the following corollary is omitted here; however, it is implicit in the proof of Lemma 3.5 below. We do not use this corollary explicitly. Nevertheless, it gives insight into the analysis below.

COROLLARY D.1. *If a subgraph H of G connects two vertices u_1, u_2 from different components of F_2 (which are contracted versions of the components in F), then H exhausts the color corresponding to at least one of u_1 and u_2.*

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

LEMMA D.2. *Let L be the set of colors exhausted by subgraph G' of G^*. The cost of $G'(V^*, E')$ is at least $\sum_{v \in L} \phi_v$.*

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

Proof. The cost of $G'(V^*, E)$ is

$$
\sum_{e \in E'} c_e \geq \sum_{e \in E'} \sum_{S : e \in \delta(S)} y_S \qquad\qquad \text{by (D.1)}
$$

$$
= \sum_S |E' \cap \delta(S)| y_S
$$

$$
\geq \sum_{S : E' \cap \delta(S) \neq \emptyset} y_S
$$

$$
= \sum_{S : E' \cap \delta(S) \neq \emptyset} \sum_{v \in S} y_{S,v}
$$

$$
= \sum_v \sum_{S \ni v : E' \cap \delta(S) \neq \emptyset} y_{S,v}
$$

$$
\geq \sum_{v \in L} \sum_{S \ni v : E' \cap \delta(S) \neq \emptyset} y_{S,v}
$$

$$
= \sum_{v \in L} \sum_{S \ni v} y_{S,v},
$$

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

$$
= \sum_{v \in L} \phi_v \qquad\qquad \text{by a tight version of (D.1).}
$$

The algorithm `PC-Clustering` has two guarantees as referenced in the body of the paper.

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

2. $\sum_i \mathsf{SteinerForest}(G, \mathcal{D}_i^*) \leq (1 + \epsilon)\mathsf{SteinerForest}(G, \mathcal{D}^*)$.