

# Approximation algorithms for prize collecting forest problems with submodular penalty functions

Yogeshwer Sharma\*

Chaitanya Swamy†

David P. Williamson‡

## Abstract

In this paper, we study the prize-collecting version of constrained forest problems with an arbitrary 0-1 connectivity requirement function and a submodular penalty function. Our framework generalizes the Prize Collecting Generalized Steiner Tree framework of Hajiaghayi and Jain [HJ06] to incorporate more general connectivity requirements and penalty functions. We generalize their primal-dual algorithm using submodular function minimization to give a 3-approximation algorithm, and devise an LP rounding algorithm with a performance guarantee of 2.54.

## 1 Introduction and related work

Over the past decade and a half, there has been a significant amount of work in the study of approximation algorithms for network design problems (see, for example, the survey of Kortsarz and Nutov [KN06]). This study has led to the revitalization of the primal-dual method for approximation algorithms and new methods in the design of approximation algorithms, such as Jain’s iterated rounding technique [Jai01].

Early in this sequence, Bienstock, Goemans, Simchi-Levi and Williamson [BGLW93] considered a problem of allowing some connectivity constraints to be violated in exchange for paying a penalty (based on a problem earlier proposed by Balas [Bal89]). In particular, they gave an approximation algorithm for the *prize-collecting Steiner tree problem* (PCST). In the PCST, we are given an undirected graph  $G = (V, E)$  with a root vertex  $r \in V$ , nonnegative costs  $c_e \geq 0$  on the edges  $e \in E$ , and nonnegative penalties  $\pi_i \geq 0$  on the vertices  $i \in V$ . The goal is to find a tree rooted at  $r$  that minimizes the sum of the costs of the edges in the tree plus the penalties of the vertices not spanned

by the tree. The PCST captures the problem of a cable company deciding how to expand its network, where the penalties represent foregone profits [JMP00]. Goemans and Williamson [GW95] later gave a primal-dual 2-approximation algorithm for the problem.

Two recent works have expanded on this theme of allowing violated connectivity constraints in exchange for a penalty. Hayrapetyan, Swamy, and Tardos [HST05] define a version of the PCST where the objective function is now the sum of the cost of the edges in the tree plus the value of a set function  $h$  on the set of unspanned vertices. They show that when  $h$  is monotone and submodular, the primal-dual PCST algorithm can be extended to produce a 2-approximation algorithm for this problem. Hajiaghayi and Jain [HJ06] consider the extension of the PCST to the *prize-collecting generalized Steiner tree* (PCGST) problem. In the generalized Steiner tree problem, we are given an undirected graph  $G = (V, E)$ , a set of pairs of vertices  $(s_1, t_1), \dots, (s_k, t_k)$ , and nonnegative costs  $c_e \geq 0$  on the edges  $e \in E$ . We must find a minimum-cost set of edges  $F$  such that each  $s_i$ - $t_i$  pair is connected by  $F$ . In the PCGST, we are additionally given nonnegative penalties  $\pi_i \geq 0$  for each pair  $(s_i, t_i)$  of vertices. The goal is to find a set of edges  $F \subseteq E$  that minimizes the sum of the cost of the edges in  $F$  plus the penalties of the pairs of vertices not connected by  $F$ . Hajiaghayi and Jain show that by applying ideas of the primal-dual PCST algorithm to a novel integer programming formulation of the problem (which they derive from a more straightforward formulation), they are able to obtain a primal-dual 3-approximation algorithm and an LP rounding 2.54-approximation algorithm for the problem. Some of the technical difficulty of their algorithm lies in determining the next dual constraint that will go tight in the dual increase phase of their algorithm.

In this paper, we continue this study in an attempt to find a general form for network design problems where violated connectivity constraints are allowed in exchange for a penalty. Our main contribution is a general model of this problem with a very general penalty function that still allows for a good approximation al-

\*Department of Computer Science, Cornell University, Ithaca, NY 14853. Email: yogi@cs.cornell.edu. Supported by NSF grant CCF-0514628.

†Dept. of Combinatorics & Optimization, University of Waterloo, Waterloo, ON N2L 3G1. Email: cswamy@math.uwaterloo.ca. Supported by NSERC grant 32760-06.

‡Department of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853. Email: dpw@cs.cornell.edu. Supported by NSF grant CCF-0514628.

gorithm. In particular, we generalize the connectivity constraints from Steiner trees (as in PCST) and generalized Steiner trees (as in PCGST) to any 0-1 connectivity requirement function. We also generalize the penalty function allowed from a monotone submodular set function (as in [HST05]) or penalties on pairs of vertices (as in [HJ06]) to a monotone submodular function whose ground elements are sets of vertices. We impose some additional properties on the function that we will discuss later. Our model captures the problems of both Hayrapetyan et al. [HST05] and Hajiaghayi and Jain [HJ06]. We show that an extension of the PCGST algorithm gives a primal-dual 3-approximation algorithm for our model. Furthermore, we give an LP rounding algorithm for the problem with performance guarantee 2.54 when the connectivity requirement function is *proper* [GW95], extending previous work of Goemans [Goe98] (on PCST) and Hajiaghayi and Jain [HJ06] (on PCGST). For simplicity, we call our problem the *prize-collecting forest problem* (PCF).

We now specify our model. Let  $f : 2^V \rightarrow \{0, 1\}$  be any function, and  $\pi : 2^V \rightarrow \mathbb{Z}_{\geq 0}$  be a penalty function whose form we will specify later; for now, note that the domain of  $\pi$  is the set of all *collections*  $\mathcal{S}$  of subsets of vertices. We consider the network design problem of finding a subset  $F$  of edges that minimizes the cost of the edges plus the penalties of subsets whose requirement is not satisfied, that is, we want a set  $F$  minimizing  $\sum_{e \in F} c_e + \pi(\{S \subseteq V : f(S) = 1, \delta(S) \cap F = \emptyset\})$ . We can model this by the following integer program:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e + \sum_{S \subseteq 2^V} \pi(\mathcal{S}) z_S & (\text{PCF-IP}) \\ \text{s.t.} \quad & \sum_{e: e \in \delta(S)} x_e + \sum_{S \subseteq 2^V: S \ni e} z_S \geq f(S) \quad \forall S \subseteq V, S \neq \emptyset \\ & x_e, z_S \in \{0, 1\} \quad \forall e \in E, \forall S \subseteq 2^V. \end{aligned}$$

This integer program is a generalization of the one introduced by Hajiaghayi and Jain. The 0-1 variable  $x_e$  denotes whether edge  $e$  is selected to be in the solution, and the 0-1 variables  $z_S$  denote the collection of sets whose connectivity requirement we decide not to fulfill. The constraint enforces that for each set  $S$  with  $f(S) = 1$ , either we choose an edge  $e \in \delta(S)$  (where  $\delta(S)$  is the set of edges with exactly one endpoint in  $S$ ) or  $z_S = 1$  for some  $\mathcal{S} \ni S$ . Our objective minimizes the cost of the edges selected plus the penalty on the collection of sets whose requirement is not satisfied.

We require the following properties from our penalty function in order for our algorithm to work:

- (Emptyset property)  $\pi(\emptyset) = 0$ ;
- (Monotonicity) If  $\mathcal{S} \subseteq \mathcal{T}$ , then  $\pi(\mathcal{S}) \leq \pi(\mathcal{T})$ .

- (Submodularity) For any collections  $\mathcal{S}$  and  $\mathcal{T}$ ,  $\pi(\mathcal{S}) + \pi(\mathcal{T}) \geq \pi(\mathcal{S} \cup \mathcal{T}) + \pi(\mathcal{S} \cap \mathcal{T})$ .
- (Union property) For any two subsets  $S_1$  and  $S_2$ ,  $\pi(\{S_1, S_2, S_1 \cup S_2\}) = \pi(\{S_1, S_2\})$ .
- (Complement property) For any subset  $S \subseteq V$ ,  $\pi(\{S, S^c\}) = \pi(\{S\})$ .
- (Inactivity property) For any subset  $S \subseteq V$  with  $f(S) = 0$ ,  $\pi(\{S\}) = 0$ .

The first property says that if we fulfill all requirements, there is no penalty. The second says that the penalty cannot go down for fulfilling fewer requirements. The third one says that the more requirements go unfulfilled, the less costly it is to violate some particular requirement. The fourth and fifth state that there is no additional penalty for violating the requirement on the union of two sets, if the two sets are already violated, or on a set if its complement is already violated. The final property states that there is no penalty associated with a set on which we have no requirement. The final three properties have some interesting connections to the notion of a 0-1 proper connectivity requirement function as defined in [GW95]. They also appear to be necessary for our algorithms and their analyses to work.

Though our algorithms are generalizations of previous work, we need some non-trivial extensions to both the algorithms and the analyses in order to obtain our results. For the primal-dual algorithm, our general penalty function necessitates a bit of care in bounding the cost of the edges. Also, implementing the primal-dual algorithm in polynomial time requires some work. We assume we have oracle access to the penalty function  $\pi$ , but note that we must be careful in using it since the input size to the oracle is potentially exponential in the size of the input to the problem. As with Hajiaghayi and Jain, finding the next dual constraint to go tight in the primal-dual algorithm is nontrivial, but we show that our model allows us to find this by submodular function minimization. Given integer penalty values and integer edge costs, we will show that our algorithm runs in time polynomial in  $\log \max_S \pi(\mathcal{S})$  and the problem input size.

For the LP rounding algorithm, the main difficulty is solving the linear program in polynomial time, since the linear program has an exponential number of constraints and a doubly-exponential number of variables. We reformulate the linear program as a convex program and carefully apply the ellipsoid method to solve it. For our separation oracle, we use Edmonds' method [Edm70] for optimizing over polymatroids associated with the submodular penalty function  $\pi$ , and show that

despite the fact that the base set is exponential, we can find a solution for this problem in polynomial time.

The rest of the paper is organized as follows. In Section 2, we discuss how the results of Hajiaghayi and Jain [HJ06] are special cases of our model, and we discuss further aspects of our class of penalty functions. In Section 3, we give our primal-dual algorithm. In Section 4, we show the primal-dual algorithm is a 3-approximation algorithm. Section 5 discusses implementation details of the primal-dual algorithm, and shows that it can be implemented in polynomial time. This is a non-trivial issue since we have to determine which of a family of  $2^{2^n}$  dual inequalities corresponding to the families  $\mathcal{S}$  will next be tight. In Section 6, we sketch our LP rounding algorithm and the algorithm for solving the linear program. Due to space constraints some proofs, and most of the LP rounding argument, are omitted.

## 2 Our model

In this section, we further discuss our prize-collecting forest problem (PCF). We begin by discussing the connection of PCF to the 0-1 proper functions of [GW95]. Next, we show that the result of Hajiaghayi and Jain [HJ06] is a special case of our model. Finally, we show that PCF does not reduce to the PCGST of Hajiaghayi and Jain. We also note that the model of Hayrapetyan et al. [HST05] is a special case of our model, but defer details to the full version of the paper.

**2.1 Connection to proper functions** *Proper* connectivity requirement functions were first defined in [GW95]. A function  $f : 2^V \rightarrow \{0,1\}$  is proper if  $f(V) = 0$ ,  $f(V - S) = f(S)$  for all  $S \subseteq V$ , and for disjoint sets  $A$  and  $B$ ,  $f(A \cup B) \leq \max(f(A), f(B))$ .

We now argue that proper functions are closely related to the penalty function we have introduced. Given an arbitrary 0-1 connectivity function  $f : 2^V \rightarrow \{0,1\}$ , we can ensure that our algorithm returns a feasible solution for the function by setting  $\pi(\mathcal{S}) = \infty$  whenever there is  $S \in \mathcal{S}$  such that  $f(S) = 1$  and  $\pi(\mathcal{S}) = 0$  otherwise. This penalty function obeys the emptyset, monotonicity, submodularity, and inactivity properties. However, in order to obey the complement property, we cannot have  $f(S) \neq f(V - S)$ , and in order to obey the union property we cannot have  $f(A) = f(B) = 0$  and  $f(A \cup B) = 1$  for disjoint  $A$  and  $B$ . Thus for our model of penalty function, if we enforce that a feasible solution is returned for the connectivity function  $f$ , that function must be proper.

**2.2 PCGST problem of Hajiaghayi and Jain** [HJ06] Hajiaghayi and Jain [HJ06] consider the the prize-collecting generalized Steiner tree prob-

lem in which there are a set of pairs of vertices  $(s_1, t_1), \dots, (s_k, t_k)$  to connect along with penalties  $\pi_i$  for not connecting the pair  $(s_i, t_i)$ . The goal is to find a set of edges  $F$  that minimizes the cost of the edges in  $F$  plus the penalties of the pairs of vertices not connected by  $F$ . We can map this problem into our framework as follows. The connectivity requirement function  $f$  for our model is the function  $f(S) = 1$  iff there exists some  $i$  such that  $|S \cap \{s_i, t_i\}| = 1$ , and the penalty  $\pi$  for a family  $\mathcal{S}$  for our model is the sum of penalties of pairs which are separated by some set in  $\mathcal{S}$ ; that is  $\pi(\mathcal{S}) = \sum_{i: \exists S \in \mathcal{S}: |S \cap \{s_i, t_i\}| = 1} \pi_i$ . The resulting penalty, as it turns out, satisfies all properties that we require from the penalty function: it is easy to see that the emptyset, inactivity, and monotonicity properties are satisfied. Hajiaghayi and Jain show that the submodularity property is satisfied (Lemma 2.3 of [HJ06]). The complement property is not hard to see:  $V - S$  separates no more pairs than  $S$ . Similarly, for any two  $S_1, S_2$ , their union  $S_1 \cup S_2$  separates no more pairs than do  $S_1$  and  $S_2$ , so the union property is satisfied.

**2.3 Some other problems that can be cast into our model** Consider the PCGST problem of Hajiaghayi and Jain with a variation on the penalty of a solution. There are  $k$  pairs of vertices with penalties  $\pi_1 = \dots = \pi_k = 1$ . The penalty of the solution is the minimum of the number of unconnected pairs and a fixed number  $l \leq k$ . It means that the penalty increases linearly with the number of unconnected pairs, but is upper bounded by  $l$ . This problem does not fit Hajiaghayi-Jain's model. We next show that the problem can be cast into our framework.

The connectivity requirement function  $f$  is same as the one described in Section 2.2. The penalty function is also the one described in the section except that it is upper bounded by  $l$ . It is easy to see that the penalty function satisfies emptyset and monotonicity properties. The union, complement, and inactivity properties also follow the same argument as in Section 2.2. For submodularity of the penalty function, we will prove the equivalent statement that for families  $\mathcal{S}$  and  $\mathcal{T} \supseteq \mathcal{S}$  and for any subset  $S \notin \mathcal{T}$ ,  $\pi(\mathcal{S} \cup \{S\}) - \pi(\mathcal{S}) \geq \pi(\mathcal{T} \cup \{S\}) - \pi(\mathcal{T})$ . If  $\mathcal{S}$  separates more than  $l$  pairs, then the inequality is trivially true. Let  $\mathcal{S}$  separate  $a \leq l$  pairs which are to be connected and  $S$  separates an additional  $b$  pairs (not already separated by  $\mathcal{S}$ ). Clearly  $\mathcal{T}$  separates  $a^+ \geq a$  pairs and  $\mathcal{T} \cup \{S\}$  separates at most  $a^+ + b$  pairs. So the left hand side of the inequality is  $\min(a + b, l) - a$ , and the right hand side is at most  $\min(a^+ + b, l) - \min(a^+, l)$ . One can easily check that  $\min(a + b, l) - a \geq \min(a^+ + b, l) - \min(a^+, l)$ . This proves the submodularity of the function  $\pi(\cdot)$ .

### 3 The primal dual algorithm

In this section, we give a primal-dual approximation algorithm to solve our problem. The LP relaxation of the integer program (PCF-IP) is:

$$\begin{aligned} \min \quad & \sum_e c_e x_e + \sum_S \pi(S) z_S && \text{(PCF-LP)} \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e + \sum_{S': S \subseteq S'} z_{S'} \geq f(S) && \text{for all } S \subseteq V, \\ & x_e, z_S \geq 0 && \text{for all } e, S. \end{aligned}$$

The dual of the LP relaxation (PCF-LP) is

$$\begin{aligned} \max \quad & \sum_{S \subseteq V} f(S) \cdot y_S && \text{(PCF-D)} \\ \text{s.t.} \quad & \sum_{S: e \in \delta(S)} y_S \leq c_e && \forall e \in E \quad \text{(type-(e))} \\ & \sum_{S: S \subseteq \mathcal{S}} y_S \leq \pi(\mathcal{S}) && \forall \mathcal{S} \subseteq 2^V \quad \text{(type-(f))} \\ & y_S \geq 0 && \forall S \subseteq V. \end{aligned}$$

Our algorithm is similar to Algorithm A in [HJ06], though it differs in how it finds the tight family of subsets and how it performs the reverse delete step. We next give the idea behind our algorithm. For its precise description, see Algorithm 1.

We introduce some terminology first. We call an edge  $e$  *tight* if its corresponding inequality (of type-(e)) in (PCF-D) holds with equality, i.e.,  $\sum_{S: e \in \delta(S)} y_S = c_e$ . Similarly, we call a family  $\mathcal{S} \subseteq 2^V$  *tight* if its corresponding inequality (of type-(f)) holds with equality, i.e.,  $\sum_{S \in \mathcal{S}} y_S = \pi(\mathcal{S})$ . For a family  $\mathcal{S} \subseteq 2^V$ , we call  $\mathcal{T}$  the *closure* of  $\mathcal{S}$  (denoted  $\text{closure}(\mathcal{S})$ ) if  $\mathcal{T} \supseteq \mathcal{S}$  and  $\mathcal{T}$  is closed under taking unions and complements (and hence intersections and set differences too).

We keep track of three types of components, (i) active components ( $\mathcal{C}_a$ )—components that need an edge out of them ( $f(S) = 1$ ) but do not have one yet, (ii) inactive components ( $\mathcal{D}_i$ )—components having  $f(S) = 0$ , and (iii) marked components ( $\mathcal{D}_m$ )—components that needed an edge out of them originally but our algorithm has decided not to have such an edge and pay the penalty; these are denoted by  $f(S) = 1 \rightarrow 0$ .

To start off, there are  $n$  components corresponding to  $n$  singleton vertices. We set them active or inactive depending on whether their connectivity requirement is 1 or 0 respectively. This is the *initialization* phase.

In the *dual-increase* phase, we uniformly increase the duals of all active subsets in  $\mathcal{C}_a$ . In this process, either another edge becomes tight or another family of subsets becomes tight. (See Section 5 for details on how to check tightness of edges and families.) If an edge  $e$

connecting two components  $C_u$  and  $C_v$  becomes tight, we add  $e$  to the set of candidate edges for the output solution, remove  $C_u$  and  $C_v$  from the family  $\mathcal{C}_a$  of active sets (if they are there), make a new component  $C_u \cup C_v$ , and set it active (and add it to  $\mathcal{C}_a$ ) if  $f(C_u \cup C_v) = 1$ , and inactive otherwise (and add it to  $\mathcal{D}_i$ ). If, on the other hand, a family  $\mathcal{S}$  of subsets  $\{S_1, S_2, \dots, S_{|\mathcal{S}|}\}$  becomes tight, we *mark* all these subsets, remove them from the family  $\mathcal{C}_a$  of active sets (if they are there), and add them to the family  $\mathcal{D}_m$  of marked sets.

When there is no active component remaining whose dual can be increased, we enter the *reverse delete* phase of our algorithm. We consider edges added to the solution in the reverse order. Consider the two components  $C_1$  and  $C_2$  resulting from deletion of edge  $e$  in the current solution. If  $e$  is the only edge out of  $C_i$  ( $i = 1$  or  $2$ ) and  $C_i$  is not in the closure of inactive sets and marked sets then we retain the edge  $e$ ; otherwise, we delete it from the solution. The idea is that if  $e$  is redundant or only necessary for sets whose penalty has been paid for (because they are either inactive or marked), it should be deleted from the solution to decrease the cost of edges in the solution.

Let  $F'$  be the set of edges remaining after the reverse delete procedure and let  $\mathcal{C}(V, F')$  be the connected components of  $F'$ . Then our algorithm returns  $F'$  as its set of edges and  $\text{closure}(\mathcal{C}(V, F'))$  as the family of subsets on which it will pay the penalty.

### 4 Analysis of the primal dual algorithm

In this section, we prove that Algorithm 1 has a performance guarantee of 3 for the prize-collecting forest problem. We first ensure that the penalty paid by the solution produced by the algorithm is no more than the cost of the optimum solution. We then show in Section 4.3 that the cost of edges finally selected by the algorithm (after reverse delete) is no more than twice the cost of the optimum solution. This will prove a performance guarantee of 3.

#### 4.1 Penalty paid by the algorithm is bounded

**by OPT** The proof in this section will break naturally into two parts. We first prove that the penalty of all marked and inactive subsets is bounded by the value of our dual solution, and follow that with the proof that the actual penalty of the algorithm is no more than the penalty of all marked and inactive subsets. We begin by inferring some properties of the penalty function.

**LEMMA 4.1.** *Let  $\mathcal{S}$  be any collection of subsets with  $S_1, S_2$  (not necessarily different subsets) in  $\mathcal{S}$ , and let  $S$  be a subset such that  $f(S) = 0$ . Then  $\pi(\mathcal{S}) = \pi(\mathcal{S} \cup \{S_1 \cup S_2\}) = \pi(\mathcal{S} \cup \{S_1^c\}) = \pi(\mathcal{S} \cup \{S\})$ .*

**Data:** A graph  $G = (V, E)$ , edge cost function  $c_e : E \rightarrow \mathbb{Q}_{\geq 0}$ , connectivity requirement function  $f : 2^V \rightarrow \{0, 1\}$ , and penalty function  $\pi : 2^V \rightarrow \mathbb{Z}_{\geq 0}$ .

**Result:**  $F' \subseteq E$  minimizing  $\sum_{e \in F'} c_e + \pi(\{S \subseteq V : f(S) = 1, \delta(S) \cap F' = \emptyset\})$ .

```

1 Let  $F = \emptyset$  (the edge set of the solution).;
2 Let  $\mathcal{C}_a = \{\{v\} | v \in V, f(\{v\}) = 1\}$  (active sets),
 $\mathcal{D}_i = \{\{v\} | v \in V, f(\{v\}) = 0\}$  (inactive sets),
 $\mathcal{D}_m = \emptyset$  (marked sets), and  $\mathcal{M} = \{\{v\} : v \in V\}$ 
(maximal components). Set  $d_v = 0$  for all  $v \in V$ 
and (implicitly) set  $y_S = 0$  for all  $S \subseteq V$ .
3 while  $\mathcal{C}_a \neq \emptyset$  do
4   Find an edge  $e = (u, v)$  with  $u \in C_u \in \mathcal{M}$ ,
 $v \in C_v \in \mathcal{M}$ ,  $C_u \neq C_v$  minimizing
 $\varepsilon_e = (c_e - d_u - d_v) / (f(C_u) + f(C_v))$ .
5   Find a non-tight family  $\mathcal{S} = \{S_1, \dots, S_{|\mathcal{S}|}\}$ 
with each  $S_i \in \mathcal{C}_a \cup \{S : y_S > 0\}$  minimizing
 $\varepsilon_f = (\pi(\mathcal{S}) - \sum_{S \in \mathcal{S}} y_S) / (\sum_{S \in \mathcal{S}} f(S))$ .
6   Set  $\varepsilon = \min\{\varepsilon_e, \varepsilon_f\}$ .
7   Set  $y_C = y_C + \varepsilon$  for all  $C \in \mathcal{C}_a$ .
8   if  $\varepsilon = \varepsilon_e$  then
9     Let  $F \leftarrow F \cup \{e = \{u, v\}\}$ ,
 $\mathcal{M} \leftarrow \mathcal{M} \cup \{C_u \cup C_v\} - \{C_u, C_v\}$ , and
 $\mathcal{C}_a \leftarrow \mathcal{C}_a \setminus \{C_u, C_v\}$ ;
10    if  $f(C_u \cup C_v) = 1$  then
11      |  $\mathcal{C}_a \leftarrow \mathcal{C}_a \cup \{C_u \cup C_v\}$ 
12    else
13      |  $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{C_u \cup C_v\}$ 
14    end
15  else
16    |  $\mathcal{D}_m \leftarrow \mathcal{D}_m \cup \{S_1, S_2, \dots, S_{|\mathcal{S}|}\}$ ,
17    |  $\mathcal{C}_a \leftarrow \mathcal{C}_a - \{S_1, S_2, \dots, S_{|\mathcal{S}|}\}$ 
18  end
19 Let  $F_l = F = \{e_1, e_2, \dots, e_l\}$ . Edges are
arranged in order they were added to  $F$ .
20 for  $j = l$  down to 1 do
21   Consider the graph  $H_j = (V_j, E_j)$  where
 $V_j = \mathcal{C}(V, \{e_1, e_2, \dots, e_{j-1}\})$  (components)
and  $E_j = F_j \setminus \{e_1, e_2, \dots, e_{j-1}\}$ . Let  $u_j(e_j)$ 
and  $v_j(e_j)$  be the two endpoints of  $e_j$  in  $H_j$ .
22   if  $\delta(u_j(e_j)) = \{e_j\} \wedge u_j(e_j) \notin \text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$ 
 $\vee \delta(v_j(e_j)) = \{e_j\} \wedge v_j(e_j) \notin \text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$ 
then
23     |  $F_{j-1} = F_j$ .
24   else
25     |  $F_{j-1} = F_j \setminus e_j$ .
26   end
27 end
28 Return  $F' = F_0$  as the set of edges.

```

**Algorithm 1:** A primal-dual algorithm for PCF.

Let  $F'$  be the final set of edges returned by the algorithm, with  $\mathcal{C}(V, F')$  the set of components of  $F'$ . Using Lemma 4.1, we note that for a given family  $\mathcal{S}$  of subsets of  $V$ ,  $\pi(\mathcal{S}) = \pi(\text{closure}(\mathcal{S}))$ . We will prove that  $\pi(\mathcal{D}_i \cup \mathcal{D}_m) \leq \sum_{S \subseteq V} y_S$ . Since the dual value is bounded above by OPT, this will prove that  $\pi(\text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)) \leq \text{OPT}$ . Then we show that  $\text{closure}(\mathcal{C}(V, F')) \subseteq \text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$ , which by the monotonicity of  $\pi$  will prove that the penalty paid by the solution returned by the algorithm is at most OPT.

The following lemma states that if two families are tight, then their union is also tight. This is the same as Corollary 2.2 in [HJ06].

**LEMMA 4.2.** *Let  $y$  be a feasible solution to dual program (PCF-D), and  $\mathcal{S}$  and  $\mathcal{T}$  be tight families w.r.t.  $y$ . Then  $\mathcal{S} \cup \mathcal{T}$  is also tight w.r.t.  $y$ .*

Since  $\mathcal{D}_m$  is a union of several tight families, it is tight by Lemma 4.2, i.e.,  $\sum_{S \in \mathcal{D}_m} y_S = \pi(\mathcal{D}_m)$ . Thus,

$$\begin{aligned} \pi(\text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)) \\ = \pi(\mathcal{D}_i \cup \mathcal{D}_m) = \pi(\mathcal{D}_m) = \sum_{S \in \mathcal{D}_m} y_S \leq \sum_{S \subseteq V} y_S. \end{aligned}$$

The second equality follows from Lemma 4.1, since the sets in  $\mathcal{D}_i$  are inactive. The bound on the penalty of the closure of marked and inactive subsets follows.

Recall that  $F'$  is the set of edges finally returned by the algorithm and  $\mathcal{C}(V, F')$  denotes the set of connected components in the graph with vertices  $V$  and edges  $F'$ . We now prove that the penalty incurred by the algorithm,  $\pi(\text{closure}(\mathcal{C}(V, F')))$ , is no more than  $\pi(\text{closure}(\mathcal{D}_i \cup \mathcal{D}_m))$ . We will prove this bound by showing that  $\mathcal{C}(V, F') \subseteq \text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$ . Since the penalty function is monotonically increasing and the closure function is idempotent, the bound on the penalty of the algorithm will follow.

Note that in Algorithm 1, line 23 and 25, we define a series of sets of edges  $F' = F_l \supseteq F_{l-1} \supseteq \dots \supseteq F_1 \supseteq F_0 = F'$ . We need to prove that  $\mathcal{C}(V, F_j) \subseteq \text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$  for all  $j = l, l-1, \dots, 1, 0$ . Since  $F' = F_0$ , this will prove the required claim. We state it as the following lemma.

**LEMMA 4.3.** *For all  $j$ ,  $\mathcal{C}(V, F_j) \subseteq \text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$ .*

This finishes the proof that the penalty paid by the algorithm is at most the cost of the optimal solution.

**4.2 The minimal augmentation property of the reverse delete step** We present another simple property of the reverse delete step, which we will need for proving the final performance guarantee.

**LEMMA 4.4.** *If  $e_j$  cannot be deleted in reverse delete step when it is  $e_j$ 's turn, then  $e_j$  cannot be deleted later keeping both endpoint components in  $\text{closure}(\mathcal{D}_i \cup \mathcal{D}_m)$ .*

The contrapositive of Lemma 4.4 tells that if we can delete  $e_j$  at some later instance than it was considered for deletion, then it could also have been deleted when it was considered. In particular,  $F_{j-1} \setminus \{e_1, e_2, \dots, e_{j-1}\}$  gives a minimal augmentation to the set  $\{e_1, e_2, \dots, e_{j-1}\}$ .

**4.3 Cost of edges** In this section, we will prove that the cost of edges in the output is no more than twice the value of the dual solution. Let  $F'$  be the final set of edges output by the algorithm. We want to show that

$$\sum_{e \in F'} c_e \leq \left(2 - \frac{2}{n}\right) \sum_{S \subseteq V} y_S. \quad (4.1)$$

Our proof follows the standard outline given in [GW95]. The only novelty is the proof that all “leaf components” must be active.

Since all the edges that we include in our solution are tight, the left hand side can be rewritten as (after changing order of summation)

$$\begin{aligned} \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S &= \sum_{S \subseteq V} y_S \cdot |F' \cap \delta(S)| \\ &\leq \left(2 - \frac{2}{n}\right) \sum_{S \subseteq V} y_S. \end{aligned} \quad (4.2)$$

We will prove this by induction on the number of iterations of our algorithm. The claim is initially true (both sides are zero).

For the induction step, let us say that this inequality holds after  $i$  steps of the algorithm and  $e_{j-1}$  was the last edge added to the solution at this point in the algorithm. Let  $H'_j$  be a graph whose vertices correspond to the connected components  $\mathcal{C}(V, \{e_1, \dots, e_{j-1}\})$  and whose edges correspond to the set  $F' \setminus \{e_1, \dots, e_{j-1}\} = F_{j-1} \setminus \{e_1, e_2, \dots, e_{j-1}\}$ ; that is, edge  $e$  in  $H'_j$  joins two vertices if the corresponding edge joins two connected components in  $\mathcal{C}(V, \{e_1, \dots, e_{j-1}\})$ . Let  $\mathcal{C}_a$  denote the set of active components in  $H'_j$  and for  $C \in \mathcal{C}(V, \{e_1, e_2, \dots, e_{j-1}\})$ , let  $d(C)$  denote the degree of  $C$  in  $H'_j$ . In the  $(i+1)$ -st iteration, if the dual variables grow by  $\varepsilon$ , then the left hand side of Equation (4.2) increases by  $\varepsilon \times \sum_{C_a \in \mathcal{C}_a} d(C_a)$  and the right hand side increases by  $\varepsilon \times \left(2 - \frac{2}{n}\right) |\mathcal{C}_a|$ . Therefore proving induction step reduces to proving

$$\sum_{C_a \in \mathcal{C}_a} d(C_a) \leq \left(2 - \frac{2}{n}\right) |\mathcal{C}_a|. \quad (4.3)$$

This is equivalent to proving that the average degree of active components contained in  $\mathcal{C}(V, \{e_1, \dots, e_{j-1}\})$  in the graph  $H'_j$  is at most  $2 - \frac{2}{n}$ . We will prove that

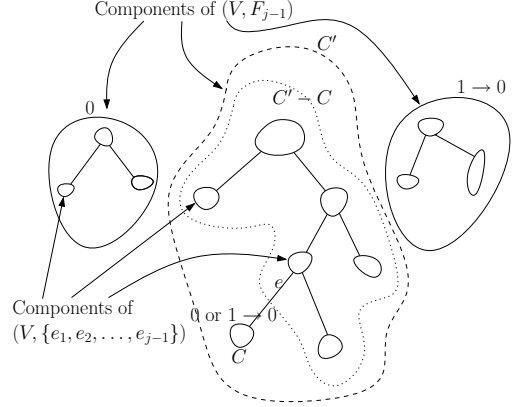


Figure 1: Illustration of the proof that there are no inactive/marked leaves.

there are no non-active leaves in  $H'_j$ , which implies, by a standard argument [GW95], that the average degree of active components is at most  $(2 - 2/n)$ .

For the sake of contradiction, assume that there is some non-active leaf  $v$  having a single incident edge  $e$  in  $H'_j$  (see Figure 1 for an example). Let  $C$  be the corresponding component; since we assume  $C$  is not active, either it is marked or inactive. In either case, it is in  $\text{closure}(\mathcal{D}_m \cup \mathcal{D}_i)$ . By Lemma 4.3, all components of  $\mathcal{C}(V, F_{j-1})$  (which are same as components of  $H'_j$  since  $F_{j-1} = \{e_1, \dots, e_{j-1}\} \cup F'$ ) are in  $\text{closure}(\mathcal{D}_m \cup \mathcal{D}_i)$ . Consider the component  $C'$  of  $\mathcal{C}(V, F_{j-1})$  such that  $C \subseteq C'$ . Then since  $C \in \text{closure}(\mathcal{D}_m \cup \mathcal{D}_i)$  and all components of  $\mathcal{C}(V, F_{j-1}) \in \text{closure}(\mathcal{D}_m \cup \mathcal{D}_i)$ , it must be the case that  $C' - C \in \text{closure}(\mathcal{D}_m \cup \mathcal{D}_i)$ . Since both  $C$  and  $C' - C$  are in  $\text{closure}(\mathcal{D}_m \cup \mathcal{D}_i)$ , by Lemma 4.4, edge  $e$  must have been deleted in the reverse delete step, which is a contradiction to the fact that it is in  $F'$ . This proves that there are no non-active leaves in  $H'_j$ , hence, the average degree of active components in  $H'_j$  is at most  $2 - 2/n$ , proving the claim and Equation (4.1).

## 5 Implementation of the algorithm

We now argue that our algorithm can be implemented in polynomial time. The central difficulty is that of finding which of the  $2^{2^n}$  type-(f) dual constraints corresponding to family of subsets will next become tight given our dual increase scheme. We will address this in Section 5.3; however, we need some preliminary discussion first.

**5.1 Polynomial time complexity** Algorithm 1 makes at most  $2n$  iterations of the while statement. This is because in each iteration of the while loop, we decrease the sum of the number of active components and the number of components. If an edge becomes tight, then we decrease the total number of components (and

may also decrease the number of active components), and if a family of subsets becomes tight, we again decrease the number of active components. To start with, we have  $n$  components, all of which can be active, which gives rise to a bound of  $2n$ .

At any point, the collection of sets that have positive dual form a laminar family (that is, for any pair of sets with positive dual, either the two are disjoint or one contains the other). This implies that at the end of the algorithm there are at most  $2n$  sets with positive dual. Each step through the while statement takes polynomial time. Checking the tightness of an edge  $e$  requires us to look at all the components with positive dual and adding the duals which have the edge  $e$  in their boundary. We will show in Section 5.3 how to check the tightness of a family of subsets via binary search and submodular function minimization, which will be implementable in polynomial time.

The reverse delete step is at most  $n$  iterations and each iteration requires us to check whether two end point components of an edge are in the closure of certain sets. A polynomial-time algorithm for checking whether a set is in the closure of certain sets is standard and we omit the details in this version of the paper. Thus the whole reverse delete step also runs in polynomial time.

**5.2 Minimal tight families** Let  $\mathcal{D}_+$  denote the family of subsets of  $V$  having non-zero dual. We will prove a property of the family of subsets which ought to go tight next in order to significantly narrow down our search for tight families from  $2^{2^n}$  candidates to  $2^{2n}$  candidates. We show that a family containing a non-active set whose dual value is zero does not need to be considered for determining a tight family of subsets.

**LEMMA 5.1.** *Let  $\mathcal{S}$  be a family of subsets of  $V$  containing a non-active subset  $S \in \mathcal{S}$  with  $y_S = 0$ . If  $\mathcal{S}$  goes tight next while we increase the dual variables of active sets, then there exists a subfamily  $\mathcal{S}'$  of  $\mathcal{S}$  that will go tight no later than  $\mathcal{S}$  goes tight.*

*Proof.* Let  $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$  be the family to go tight next the earliest,  $S_k$  be a non-active subset such that  $y_{S_k} = 0$ , and the first  $j$  subsets of  $\mathcal{S}$  (without loss of generality) be active sets where  $1 \leq j \leq k-1$ . We show that the family  $\mathcal{S}' = \mathcal{S} - \{S_k\}$  goes tight no later than  $\mathcal{S}$ . Indeed, the increase in the dual variables needed for  $\mathcal{S}'$  to go tight is  $\varepsilon_{\mathcal{S}'} = \frac{1}{j} \left( \pi(\mathcal{S}') - \sum_{i=1}^{k-1} y_{S_i} \right)$  and the increase in the dual variables needed for  $\mathcal{S}$  to go tight is  $\varepsilon_{\mathcal{S}} = \frac{1}{j} \left( \pi(\mathcal{S}) - \sum_{i=1}^k y_{S_i} \right)$ . Using monotonicity of the function  $\pi$  and the fact that  $y_{S_k} = 0$ , we conclude that  $\varepsilon_{\mathcal{S}'} \leq \varepsilon_{\mathcal{S}}$  proving that the family  $\mathcal{S}'$  goes tight no later than the family  $\mathcal{S}$ . ■

Therefore, when looking for which family becomes tight next, we only need to consider subfamilies of active sets and sets with positive duals (subfamilies of  $\mathcal{C}_a \cup \mathcal{D}_+$ ). We now proceed to present a procedure to find which family of subsets goes tight next.

**5.3 The method to find a tight family** We want to find the maximum value of  $\varepsilon$  such that if we increase all active duals by  $\varepsilon$ , all constraints of type-(f) in the dual program are still satisfied. We can replace “all constraints of type-(f)” to “all constraints of type-(f) corresponding to subfamilies of  $\mathcal{C}_a \cup \mathcal{D}_+$ ” in the above statement, since the first constraint to be violated as we increase dual variables will correspond to a subfamily of  $\mathcal{C}_a \cup \mathcal{D}_+$  as proved in Lemma 5.1. This problem can be written in the form of the following program:

$$\begin{aligned} & \text{maximize } \varepsilon, \\ & \text{s.t. } \sum_{S \in \mathcal{S}} y_S + \varepsilon |\text{active}(\mathcal{S})| \leq \pi(\mathcal{S}); \quad \forall \mathcal{S} \subseteq \mathcal{C}_a \cup \mathcal{D}_+. \end{aligned}$$

Here,  $\text{active}(\mathcal{S})$  denotes active sets contained in  $\mathcal{S}$ . Let  $\varepsilon_0$  be the optimum value of  $\varepsilon$  above. We use binary search to find the correct value of  $\varepsilon_0$ . We keep a lower bound  $\varepsilon_{low}$  on the value of  $\varepsilon_0$  (take  $\varepsilon_{low} = 0$  to start with), and an upper bound  $\varepsilon_{high}$  on the value of  $\varepsilon_0$  (take  $\varepsilon_{high} = \frac{\pi(\mathcal{S}) - \sum_{S \in \mathcal{S}} y_S}{|\text{active}(\mathcal{S})|}$  for a particular family to start with, say  $\frac{\pi(\{S\}) - y_S}{1}$  for some  $S \in \mathcal{C}_a$ ).

Since  $\varepsilon_0 \in [\varepsilon_{low}, \varepsilon_{high}]$ ,  $\varepsilon_0 \in [\varepsilon_{low}, \varepsilon_{middle}]$  or  $\varepsilon_0 \in [\varepsilon_{middle}, \varepsilon_{high}]$  where  $\varepsilon_{middle} = (\varepsilon_{low} + \varepsilon_{high})/2$ . To determine which interval  $\varepsilon_0$  lies in, we use submodular function minimization. If  $\varepsilon_0 \geq \varepsilon_{middle}$ , then the minimum of the function  $\pi'(\mathcal{S}) \stackrel{\text{def}}{=} \pi(\mathcal{S}) - \sum_{S \in \mathcal{S}} y_S - \varepsilon_{middle} |\text{active}(\mathcal{S})|$  over all  $\mathcal{S} \subseteq \mathcal{C}_a \cup \mathcal{D}_+$  is nonnegative. If  $\varepsilon_0 < \varepsilon_{middle}$ , then the minimum of the function  $\pi'(\mathcal{S})$  over all subfamilies of  $\mathcal{C}_a \cup \mathcal{D}_+$  is strictly negative (because some constraint is violated). Also note that both the conditions above are if and only if conditions. Therefore testing whether  $\varepsilon_0 \geq \varepsilon_{middle}$  reduces to checking whether the minimum of the function  $\pi'(\mathcal{S})$  over subfamilies of  $\mathcal{C}_a \cup \mathcal{D}_+$  is nonnegative. The function  $\pi'(\mathcal{S})$  is a submodular function for a fixed value of  $\varepsilon_{middle}$  since its first term is submodular and the last two terms are modular. We apply a polynomial-time submodular function minimization algorithm [IFF01, Sch00] to minimize the function and hence determine which of the two interval  $\varepsilon_0$  lies in. Note that we can do this in oracle polynomial-time and time polynomial in  $n$ , since the ground set of  $\pi'$  are the sets in  $\mathcal{C}_a \cup \mathcal{D}_+$ , and we have previously argued that there are at most  $2n$  such sets. This further implies that we need at most  $O(n^2)$  space to query the oracle for  $\pi(\cdot)$ .

Once our interval  $[\varepsilon_{low}, \varepsilon_{high}]$  becomes “sufficiently”

small, we stop the binary search and find the right value of  $\varepsilon_0$  in that interval. This is the intuitive idea, which it turns out takes a bit of care to turn into a formal proof. We need the following lemma to start.

**LEMMA 5.2.** *The value of each non-zero dual variable  $y_S$  and each amount  $\epsilon$  of dual increase in each iteration of the algorithm can be expressed by rational numbers whose numerators are integers of value at most  $(\max_S \pi(\mathcal{S}) + \max_{e \in E} c_e) \cdot (6n^2)!$  and whose denominators are integers of value at most  $(6n^2)!$ . Thus these variables can be expressed with bit complexity at most  $O(\log(\max_S \pi(\mathcal{S})) + \log \max_{e \in E} c_e + n^2 \log n)$ .*

If we do binary search until the window size  $|\varepsilon_{high} - \varepsilon_{low}|$  is smaller than  $(1/(6n^2)!)^2$ , there can be at most one rational number in the window with denominator at most  $(6n^2)!$ . We now want to determine the exact value of  $\varepsilon$  in this interval. We use a procedure Best Approximation from Grötschel, Lovász, and Schrijver [GLS88, Theorem 5.1.9], which determines a rational number  $\beta$  of denominator at most  $N$  that minimizes  $|\alpha - \beta|$  for input  $\alpha$ . It does so in time polynomial in the encoding size of  $N$  and  $\alpha$ . We run one more submodular minimization to halve the window size of the binary search, then apply this procedure with  $N = (6n^2)!$  and  $\alpha$  set to whichever of  $\varepsilon_{high}$  or  $\varepsilon_{low}$  was last modified. This guarantees that Best Approximation will find  $\varepsilon_0$ .

Therefore, finding the right value of  $\varepsilon_0$  can be done in time polynomial in  $\log(\max_S \pi(\mathcal{S})) + \log(\max_e c_e) + n^2 \log n$ . The number of iterations of binary search required to bring the interval length down to  $1/((6n^2)!)^2$  is at most  $O(\log(\max_S \pi(\mathcal{S})) + n^2 \log n)$ . Furthermore, each iteration of binary search requires polynomial time, so the overall time of finding a tight family can be bounded by a polynomial in  $\log(\max_S \pi(\mathcal{S})) + \log(\max_e c_e) + n^2 \log n$ .

Once we find the right value of  $\varepsilon_0$ , we solve the problem of minimizing  $\pi(\mathcal{S}) - \sum_{S \in \mathcal{S}} y_S - \varepsilon_0 |\text{active}(\mathcal{S})|$  over subfamilies of  $\mathcal{C}_a \cup \mathcal{D}_+$ . As argued above, the minimum value will be zero and the corresponding minimizer family will be the candidate family to go tight next. Thus, the family to go tight next can be found in polynomial time.

## 6 An LP-rounding approach

In this section, we present an LP-rounding algorithm with a better performance guarantee when the connectivity requirement function  $f$  is proper. Notice that formulation (PCF-LP) has an exponential number of constraints and a doubly exponential ( $2^{2^n}$  where  $n = |V|$ ) number of variables, so even solving the LP is a challenging task. In fact, it is not even clear that a basic solution of (PCF-LP) admits a polynomial-size descrip-

tion — a basic solution of the LP may yet set  $2^n$  of the  $z_S$  variables to be positive!

Our result is built on three components. First, we show that by suitably transforming (PCF-LP), one can obtain a *compact, convex programming relaxation* of the problem that is equivalent to (PCF-LP) and has only  $x_e$  variables. Next, we give a simple procedure to round any fractional solution to this convex program to an integer solution losing a factor of at most 2.54. Since the convex program is equivalent to the LP (PCF-LP), the rounding procedure may also be viewed as a rounding procedure for (PCF-LP) with the property that one only needs the  $x$ -component of the fractional solution in order to implement it. Finally, we show that the convex program can be solved efficiently using the ellipsoid method. The key ingredient that we need to run the ellipsoid method is a procedure that computes a subgradient of the objective function at any point  $x$ . Although the convex program yields a compact relaxation of the prize-collecting problem, this poses several challenges since computing the subgradient at a given point requires solving an exponential-size linear program. Nevertheless, we will prove certain structural properties and show that a subgradient can indeed be computed in polynomial time.

### 6.1 A compact convex programming relaxation

The idea behind the compact formulation is simple. The idea behind the compact formulation is simple. Suppose we fix the values of the  $x_e$  variables. Then, one can consider an optimal setting of the  $z_S$  variables corresponding to these  $x_e$  values and denote the penalty incurred for these  $z_S$  values (which is a function of only the  $x_e$  variables) by  $g(x)$ . So we can restate the objective function as  $\sum_e c_e x_e + g(x)$ , and our problem is to minimize  $\sum_e c_e x_e + g(x)$  subject to the constraints that  $0 \leq x_e \leq 1$  for every edge  $e$ . More formally, we obtain the following program:

$$\min \quad h(x) := \sum_e c_e x_e + g(x) \quad (\text{PC-CP})$$

$$\text{subject to} \quad 0 \leq x_e \leq 1 \quad \forall e,$$

$$\text{where} \quad g(x) := \min \sum_S \pi(\mathcal{S}) z_S \quad (\text{Pen-P})$$

$$\text{s.t.} \quad \sum_{S: S \in \mathcal{S}} z_S \geq f(S) - \sum_{e \in \delta(S)} x_e \quad \forall S \subseteq V,$$

$$z_S \geq 0 \quad \forall S.$$

Here (Pen-P) is the LP that determines the penalty incurred at a fractional solution  $x$ . It is easy to see that (PC-CP) is equivalent to (PCF-LP), and it is straightforward to show that the objective function of (PC-CP) is convex. Let  $\text{OPT}_{LP}$  denote the optimal value of (PC-CP) (and (PCF-LP)).

**6.2 A rounding procedure** We now describe a procedure for rounding a fractional solution to (PC-CP), based on the rounding procedure given by Bienstock et al. [BGSLW93] and Goemans [Goe98] for the prize-collecting TSP and Steiner tree problems. We assume that the requirement function  $f$  is a proper function.

Let  $x$  be a fractional solution to (PC-CP). For a set of edges  $E'$ , we use  $x(E')$  to denote  $\sum_{e \in E'} x_e$ . Let  $\alpha \in (0, 1)$  be a parameter that we will set later. For any collection  $\mathcal{S} \subseteq 2^V$ , define  $\text{z-clos}(\mathcal{S})$  to be the maximal collection  $\mathcal{T} \supseteq \mathcal{S}$  such that  $\pi(\mathcal{T}) = \pi(\mathcal{S})$ . Note that  $\mathcal{T}$  is well-defined (i.e., the maximal collection is unique), and  $\mathcal{T} \supseteq \text{closure}(\mathcal{S})$ . We need the following structural property (whose proof we leave to the full version of the paper): one can compute in polynomial time a *laminar family* of sets  $T_1, \dots, T_k$ ,  $k \leq n$ , with  $f(T_i) = 1$  for each  $i$ , where the sets are ordered so that  $w_0 := 0 \leq w_1 := x(\delta(T_1)) \leq \dots \leq w_k := x(\delta(T_k)) \leq w_{k+1} := 1$ , such that if  $\mathcal{S}_i = \{T_1, \dots, T_i\}$  for  $i = 1, \dots, k$  with  $\mathcal{S}_0 = \emptyset$ , and  $\mathcal{T}_i = \text{z-clos}(\mathcal{S}_i)$  for  $i = 0, \dots, k$ , then the solution  $z_{\mathcal{T}_i} = w_{i+1} - w_i$  for  $i = 0, \dots, k$ , and  $z_{\mathcal{S}} = 0$  for any other  $\mathcal{S} \subseteq 2^V$ , is an optimal solution to (Pen-P) at the point  $x$ . Moreover,  $\mathcal{T}_k = 2^V$ .

So suppose that we are given sets  $T_1, \dots, T_k$  with the above properties, and let  $z$  be the optimal solution to (Pen-P) determined by these sets. For any set  $S \subseteq V$ , let  $i(S) \in \{0, \dots, k\}$  be the smallest index such that  $\pi_{\mathcal{S}_i}(S) \equiv \pi_{\mathcal{S}_i}(\{S\}) := \pi(\mathcal{S}_i \cup \{S\}) - \pi(\mathcal{S}_i) = 0$ . Equivalently  $i(S)$  is the smallest index such that  $S \in \mathcal{T}_i$ . Observe that such an index always exists. Consider the function  $\hat{f}: 2^V \mapsto \{0, 1\}$  defined by  $\hat{f}(S) = 1$  if  $f(S) = 1$  and  $\sum_{i \geq i(S)} z_{\mathcal{T}_i} < \alpha$ , and 0 otherwise.

One can argue that  $\hat{f}$  is a proper function. Note that  $\hat{f}(S)$  can be calculated in polynomial time for any set  $S$  (given an oracle for the function  $f$ ). Thus, one can use the Goemans-Williamson (GW) primal-dual algorithm [GW95] to obtain an *integer* feasible solution  $\tilde{x}$  to the following network design problem determined by the connectivity function  $\hat{f}$ :

$$\min \left\{ \sum_e c_e u_e \text{ s.t. } u(\delta(S)) \geq \hat{f}(S) \quad \forall S \subseteq V, \right. \\ \left. u_e \geq 0 \quad \forall e \in E. \right\} \quad (\text{ND-P})$$

We return  $\tilde{x}$  as the final solution.

We now sketch the analysis. Let  $\mathcal{P} = \{S \subseteq V : f(S) = 1, \tilde{x}(\delta(S)) = 0\}$ . Lemma 6.1 states that  $\hat{f}$  is a proper function. We then proceed to bound the cost of the edges bought by  $\tilde{x}$  (Lemma 6.2), and the penalty incurred for the collection  $\mathcal{P}$  (Lemma 6.3).

LEMMA 6.1. *If  $f$  is a proper function, then so is  $\hat{f}$ .*

LEMMA 6.2. *We have  $\sum_e c_e \tilde{x}_e \leq \frac{2}{1-\alpha} \cdot \sum_e c_e x_e$ .*

*Proof.* We will show that  $\frac{1}{1-\alpha} \cdot x$  is a feasible solution to (ND-P). Since the GW algorithm is a 2-approximation algorithm, this implies that  $\sum_e c_e \tilde{x}_e \leq \frac{2}{1-\alpha} \cdot \sum_e c_e x_e$ .

For any set  $S$  with  $\hat{f}(S) = 1$ , we have  $x(\delta(S)) \geq 1 - \sum_{\mathcal{S}: \mathcal{S} \in \mathcal{P}} z_{\mathcal{S}} = 1 - \sum_{i: \mathcal{S} \in \mathcal{T}_i} z_{\mathcal{T}_i} > 1 - \alpha$ . The first inequality follows since  $z$  is a feasible solution to (Pen-P) at the point  $x$ , and the last one follows since  $\sum_{i: \mathcal{S} \in \mathcal{T}_i} z_{\mathcal{T}_i} = \sum_{i \geq i(\mathcal{S})} z_{\mathcal{T}_i} < \alpha$  because  $\hat{f}(S) = 1$ . ■

LEMMA 6.3. *For each set  $S \in \mathcal{P}$ , we have  $\sum_{\mathcal{S}: \mathcal{S} \in \mathcal{P}} z_{\mathcal{S}} = \sum_{i \geq i(\mathcal{S})} z_{\mathcal{T}_i} \geq \alpha$ . Consequently, the penalty incurred,  $\pi(\mathcal{P})$ , is at most  $\frac{1}{\alpha} \cdot \sum_{\mathcal{S}} \pi(\mathcal{S}) z_{\mathcal{S}} = \frac{1}{\alpha} \cdot g(x)$ .*

THEOREM 6.1. *For any  $\alpha > 0$ , the above algorithm returns a solution of cost at most  $\frac{2}{1-\alpha} \cdot \sum_e c_e x_e + \frac{1}{\alpha} \cdot g(x)$ . Taking  $x$  to be an optimal solution to (PC-CP) and, (i) setting  $\alpha = \frac{1}{3}$  yields a 3-approximation algorithm; (ii) choosing  $\alpha \in (0, \beta]$  uniformly at random, where  $\beta = 1 - e^{-1/2}$ , yields a solution of expected cost at most  $\frac{1}{1-e^{-1/2}} \cdot \text{OPT}_{LP} \approx 2.54 \cdot \text{OPT}_{LP}$ .*

*Proof.* The proof of the first statement, which directly implies part (i), follows from Lemma 6.2 and Lemma 6.3. For part (ii), we need a more refined analysis. The expected cost of the edges is at most  $\mathbb{E} \left[ \frac{2}{1-\alpha} \right] \sum_e c_e x_e = \frac{2}{\beta} \ln(1/(1-\beta)) \sum_e c_e x_e$ . The penalty incurred for a given value of  $\alpha$  is  $\pi(\{S \subseteq V : \sum_{\mathcal{S}: \mathcal{S} \in \mathcal{P}} z_{\mathcal{S}} \geq \alpha\})$ . This is equal to  $\pi(\mathcal{T}_i)$  if  $\alpha \in (\sum_{j: i < j \leq k} z_{\mathcal{T}_j}, \sum_{j: i \leq j \leq k} z_{\mathcal{T}_j}]$  for  $i = 0, \dots, k$ , which happens with probability at most  $z_{\mathcal{T}_i}/\beta$ . Therefore, the expected penalty incurred is at most  $\frac{1}{\beta} \cdot \sum_i \pi(\mathcal{T}_i) z_{\mathcal{T}_i} = \frac{1}{\beta} \cdot g(x)$ . Thus, the total expected cost is at most  $\frac{1}{\beta} \max(2 \ln(1/(1-\beta)), 1) \cdot \text{OPT}_{LP} = \frac{1}{1-e^{-1/2}} \cdot \text{OPT}_{LP} \approx 2.54 \cdot \text{OPT}_{LP}$ . This randomized algorithm can be easily derandomized. Since there are only  $k+1$  “combinatorially-distinct” values of  $\alpha$ , one can simply try out all these values and return the least-cost solution found. ■

**6.3 Solving the convex program** The convex program (PC-CP) can be solved efficiently using the ellipsoid method provided that one can find a *subgradient* of the objective function at any given point. A subgradient of  $h$  at  $x$  is a vector  $d \in \mathbb{R}^m$ ,  $m = |E|$ , such that  $h(x') - h(x) \geq d \cdot (x' - x)$  for any feasible  $x'$ . It turns out that a subgradient can be computed if we can solve the following dual of (Pen-P).

$$\max \sum_{\mathcal{S}} (f(\mathcal{S}) - x(\delta(\mathcal{S}))) y_{\mathcal{S}} \quad (\text{Pen-D}) \\ \text{s.t.} \quad \sum_{\mathcal{S} \in \mathcal{P}} y_{\mathcal{S}} \leq \pi(\mathcal{S}) \quad \text{for all } \mathcal{S} \subseteq 2^V, \quad (6.4) \\ y_{\mathcal{S}} \geq 0 \quad \text{for all } \mathcal{S} \subseteq V.$$

But this dual has  $2^n$  variables and  $2^{2^n}$  constraints! One helpful and important fact is that since  $\pi$  is submodular and the feasible region of (Pen-D) is the *polymatroid* associated with  $\pi$ , we can use the Edmonds’ greedy algorithm [Edm70] to obtain an optimal solution to both (Pen-D) and (Pen-P). This still does not solve the problem fully since the ground set of the polymatroid is still of size  $2^n$ , and the naive greedy algorithm iterates through all ground elements. The main idea, now, is to show that there exists an optimal solution to (Pen-P) with at most  $n$  non-zero  $z$  values and an optimal solution to (Pen-D) with at most  $n$  non-zero  $y$  values, and in fact, one where the sets  $S$  with  $y_S > 0$  form a sparse *laminar collection*. Using the solution  $y$ , we can compute a subgradient of  $h$  in (PC-CP) in polynomial time, and thereby implement the ellipsoid method. Note that we do not require  $f$  to be proper in order to solve (PC-CP); this is only needed for the rounding argument.

The algorithm to compute an optimal dual solution is as follows. For a set  $S$  and collection  $\mathcal{A}$ , let  $\text{confl}(S, \mathcal{A})$  denote the number of “conflicts” between  $S$  and  $\mathcal{A}$ , that is,  $|\{S' \in \mathcal{A} : S \cap S' \neq \emptyset, S \setminus S' \neq \emptyset, S' \setminus S \neq \emptyset\}|$ . We initialize  $y_S \leftarrow 0$  for all sets  $S$ , and  $\mathcal{L} \leftarrow \emptyset$ . While  $\text{closure}(\mathcal{L}) \neq 2^V$ , (i) let  $v_{\mathcal{L}} := \min\{x(\delta(S)) : S \notin \text{closure}(\mathcal{L}), \text{confl}(S, \mathcal{L}) = 0\}$ ; find a set  $T \notin \text{closure}(\mathcal{L})$  such that  $\text{confl}(T, \mathcal{L}) = 0$  and  $x(\delta(T)) = v_{\mathcal{L}}$ . (ii) Set  $y_T = \pi_{\mathcal{L}}(T) = \pi_{\text{closure}(\mathcal{L})}(T)$  and  $\mathcal{L} \leftarrow \mathcal{L} \cup \{T\}$ .

We very briefly argue the correctness of the algorithm; the details are deferred to the full version of the paper. One way to prove this is to show that the above algorithm is in a sense equivalent to a run of Edmonds’ greedy algorithm. The above algorithm considers sets in increasing order of  $x(\delta(S))$  value, instead of decreasing  $f(S) - x(\delta(S))$  value as required by the greedy algorithm. But one can show that ordering sets in this way, and setting their values as in the greedy algorithm (so as to satisfy (6.4)), also works. Intuitively, this is because a set  $S$  with  $f(S) = 0$  adds 0 value to the penalty of any collection, so these sets can be “swapped around” without affecting optimality. Next, one can show (Lemma 6.4) that although the algorithm only considers sets that preserve the laminarity of  $\mathcal{L}$ , there is an underlying valid ordering of the sets (by increasing  $x(\delta(\cdot))$  value) that yields the same solution  $y$ . Combining these two properties shows that the solution constructed  $y$  is optimal to (Pen-D).

**LEMMA 6.4.** *For every set  $T$ , we have  $T \in \text{closure}(\mathcal{L}_T)$ , where  $\mathcal{L}_T = \{S \in \mathcal{L} : x(\delta(S)) \leq x(\delta(T))\}$ .*

Let  $\mathcal{L} = \{S_1, \dots, S_\ell\}$ ,  $\ell \leq n$ , where  $x(\delta(S_1)) \leq \dots \leq x(\delta(S_\ell))$ . Using this laminar family, one can construct a sparse optimal solution to (Pen-P) that satisfies the properties stated in Section 6.2. Let  $\mathcal{L}_{>0} =$

$\{S \in \mathcal{L} : y_S > 0\} = \{T_1, \dots, T_k\}$ , where  $T_j = S_{i(j)}$  and  $i(0) := 0 < i(1) \leq \dots \leq i(k) < i(k+1) := \ell + 1$ . Let  $\mathcal{S}_i = \{S_1, \dots, S_i\}$  for  $i = 0, \dots, \ell$  (with  $\mathcal{S}_0 := \emptyset$ ). Define  $\mathcal{S}_0 = \emptyset = T_0$ , and let  $T_j = \text{z-clos}(\mathcal{S}_{i(j)})$ ,  $w_j := x(\delta(T_j))$  for  $j = 0, \dots, k$ , and  $w_{k+1} := 1$ .

**THEOREM 6.2.** (i)  $y$  is an optimal solution to (Pen-D). (ii) The solution  $z = (z_S)_{S \subseteq 2^V}$  given by  $z_{T_j} = (w_{j+1} - w_j)$  for  $j = 0, \dots, k$ , and  $z_S = 0$  for any other  $S \subseteq 2^V$  is an optimal solution to (Pen-P).

## 7 Acknowledgments

The first author would like to thank Ara Hayrapetyan and Zoya Svitkina for useful discussions.

## References

- [Bal89] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.
- [BGSLW93] Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David P. Williamson. A note on the prize collecting traveling salesman problem. *Math. Programming*, 59:413–420, 1993.
- [Edm70] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications*, pages 68–87, 1970.
- [GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Verlag, 1988.
- [Goe98] Michel Goemans, 1998. Personal communication.
- [GW95] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.
- [HJ06] M.T. Hajiaghayi and K. Jain. The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In *SODA*, pages 631–640, 2006.
- [HST05] Ara Hayrapetyan, Chaitanya Swamy, and Éva Tardos. Network design for information networks. In *SODA*, pages 933–942, 2005.
- [IFF01] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.
- [Jai01] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.
- [JMP00] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting Steiner tree problem: theory and practice. In *SODA*, pages 760–769, 2000.
- [KN06] G. Kortsarz and Z. Nutov. Approximating minimum cost connectivity problems. In T. Gonzales, editor, *Handbook of Approximation Algorithms and Metaheuristics*. CRC Press, 2006.
- [Sch00] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Comb. Theory, Ser. B*, 80(2):346–355, 2000.