

A Technique for Improving Approximation Algorithms for Prize-Collecting Problems

Aaron Archer* MohammadHossein Bateni[†] MohammadTaghi Hajiaghayi[‡]
Howard Karloff[§]

November 17, 2008

Abstract

We study the prize-collecting versions of the Steiner tree (PCST) and traveling salesman (PCTSP) problems: given a graph (V, E) with costs on each edge and a penalty on each node, the goal is to find a tree (for PCST) or cycle (for PCTSP), that minimizes the sum of the edge costs in the tree/cycle and the penalties of the nodes not spanned by it. In addition to being a useful theoretical tool for helping to solve other optimization problems, PCST has been applied fruitfully to the optimization of real-world telecommunications networks. The most recent improvements for these problems, giving a 2-approximation algorithm for each, appeared first in 1992. The natural linear programming (LP) relaxation of PCST has an integrality ratio of 2, which has been a barrier to further improvements for this problem.

We present $(2 - \epsilon)$ -approximation algorithms for both problems, connected by a general technique for improving prize-collecting algorithms that manages to circumvent the integrality ratio barrier.

*AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. Email: aarcher@research.att.com

[†]Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540 Email: mbateni@cs.princeton.edu

[‡]AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. Email: hajiagha@research.att.com

[§]AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. Email: howard@research.att.com

1 Introduction

Prize-collecting problems involve situations where there are various demand points that desire to be “served” by some structure and we must find the structure of lowest cost to accomplish this. However, if some of the demand points are too expensive to serve, then we can refuse to serve them and instead pay a penalty. The two most famous prize-collecting problems are the prize-collecting Steiner tree (PCST) and prize-collecting traveling salesman (PCTSP) problems. In both cases, we are given a connected graph $G = (V, E)$, a non-negative cost c_e for each edge $e \in E$, denoting the cost to purchase that edge, and a non-negative *penalty* (sometimes called a *prize*) $\pi(v)$ for each node $v \in V$. In PCST, we must select some set of nodes S (possibly a singleton) to span, and a tree T spanning S so as to minimize the combined cost

$$\sum_{e \in T} c_e + \sum_{v \in \bar{S}} \pi(v) = \text{cost}(T) + \pi(\bar{S})$$

where $\bar{S} = V - S$, and we obey the usual convention that $\pi(S) = \sum_{v \in S} \pi(v)$. That is, we aim to minimize the total cost of the spanning tree plus the penalties of the nodes not spanned. PCTSP is the same, except that T should be a cycle. In PCTSP, it is typically assumed that the graph is complete and the edge costs satisfy the triangle inequality, because this opens the door to algorithms that use “shortcutting.” In PCST, these extra assumptions have no effect. When $\pi(v) = \infty$ for all $v \in V$, PCTSP simplifies to the ordinary traveling salesman problem because we have no choice but to visit all nodes. Similarly, when $\pi(v) = \infty$ on a set of terminals and $\pi(v) = 0$ on a set of Steiner nodes, PCST simplifies to an instance of the ordinary Steiner tree problem. These are both notorious NP-hard problems, so PCST and PCTSP are NP-hard as well. Thus, we are interested in obtaining approximation algorithms for them.

The first approximation algorithms for these problems were given by Bienstock et al. [6], although the PCTSP had been introduced earlier by Balas [5]. 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 [13] 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[1]. The conference version of their paper [12] appeared in 1992, and there have been no improved approximation algorithms invented for either problem, until now. In this paper, we give $(2 - \epsilon)$ -approximation algorithms for both problems.

Although the value of ϵ that we achieve is small (less than 0.01 in both cases), this is a large conceptual breakthrough, since the factor 2 was thought to be a barrier. The natural LP relaxation for PCST used in [6, 13] is known to have an integrality gap of 2, even for the ordinary Steiner tree problem, and hence cannot by itself provide a strong enough lower bound on OPT to prove an approximation factor better than 2.

PCST and PCTSP are two of the classic optimization problems, deserving of study in their own right. But even beyond that, work on the PCST has had a large impact, both in theory and in practice. At AT&T, PCST code has been used in large-scale studies aimed at optimizing access network design, both as described in Minkoff, Johnson and Phillips [15], and in later unpublished applied work that has involved the first author of the present paper.

The impact of PCST within approximation algorithms is also far-reaching. As noted by Chudak et al. [8], the PCST is a Lagrangian relaxation of the k -MST problem, which asks for the minimum-cost tree spanning at least k nodes. Moreover, they note that the PCST algorithm of Goemans and Williamson (henceforth called GW) is not just a 2-approximation algorithm, but also a *Lagrangian preserving* 2-approximation (we will give a formal definition shortly). This implies the interesting and useful property that the total edge cost of the tree T returned by GW is no more than twice the edge cost of the cheapest tree that captures at least $\pi(V_T)$ prize (i.e., pays at most $\pi(\bar{V}_T)$ penalty, where V_T is the set of nodes spanned by T). Thus,

if one runs PCST with all penalties set to some constant, and it happens to return a tree spanning k -nodes, then this tree is a 2-approximate k -MST. This property has been used in a sequence of papers [10, 4, 3] culminating in a 2-approximation algorithm for k -MST by Garg [11]. It has also been used to improve and speed up approximation algorithms for the *minimum latency problem* [2, 7]. The technique of applying Lagrangian relaxation to a problem with hard constraints, and using a Lagrangian-preserving approximation algorithm on the relaxed problem to recover an approximation algorithm for the original problem, has been successfully applied to the k -median and *uncapacitated facility location* problems as well, in a sequence of papers starting with [14].

In this paper, we use the Lagrangian-preserving guarantee of the GW algorithm in a different way. Tree T is a Lagrangian-preserving 2-approximate solution if

$$\sum_{e \in T} c_e + 2\pi(\bar{V}_T) \leq 2OPT, \tag{1}$$

whereas, to be an ordinary 2-approximation, it would suffice to satisfy the above inequality but with the 2 on the LHS changed to be 1. One way of looking at this is that GW essentially gets a 1-approximation on the penalty term, and a 2-approximation on the tree term. Our algorithm combines this observation with the fact that there are algorithms for the ordinary Steiner tree problem that achieve a factor strictly less than 2 (such as that of Robins and Zelikovsky [16], which gives a factor of less than 1.55). We can show that if either GW or OPT pays a large amount of penalty, then GW does better than a 2-approximation. In the case where both GW and OPT pay a small penalty, then, intuitively, GW is finding approximately the correct set of nodes to span, but it just might have trouble constructing the tree. We take advantage of this by using GW to help us identify a set of nodes to span, then running the Robins-Zelikovsky (RZ) algorithm using those nodes as terminals. We use RZ here as a black box. For the purposes of achieving a $(2 - \epsilon)$ factor for PCST, any ρ -approximation for ordinary Steiner tree with $\rho = 2 - \epsilon'$ would suffice.

The key to this whole scheme is Lemma 5. Where we apply this lemma, we have already established that the set of terminals on which we run RZ captures the bulk of the prize (i.e., the penalty term in the objective is not of concern). The content of the lemma is that there exists a tree spanning these nodes whose cost is not much larger than OPT . Thus, when we run RZ on this set of nodes to find a 1.55-approximation to this tree, it will give better than a 2-approximation, even after we add in the penalty term.

One novel feature of our analysis is that we use the details of the GW moat-growing algorithm to prove in Lemma 5 that there *exists* an inexpensive set of edges satisfying certain properties, but not to find it. All of the other applications of GW of which we are aware use it purely in an algorithmic sense to identify edges; ours may be the first instance where it has been utilized to provide an existence proof.

The rest of the paper is structured as follows. In Section 2 we describe our algorithm for PCST, prove some of the easier lemmas, and show how the lemmas fit together to yield our main theorem, showing that our algorithm is a $(2 - \epsilon)$ -approximation. In Section 2.1 we review the inner workings of the GW algorithm, for use in Section 2.2, where we prove the more difficult lemmas. In Section 16 we show how to apply the same techniques to derive a $(2 - \epsilon)$ -approximation algorithm for the PCTSP.

2 Prize-collecting Steiner Tree Problem

Our algorithm is relatively simple to describe, though its analysis is involved. We produce two solutions to the problem based on the GOEMANS-WILLIAMSON (GW) algorithm for PCST and the ROBINS-ZELIKOVSKY (RZ) algorithm for Steiner tree, and output the better one. Let \mathcal{I} be our input instance in which we are given a graph $G(V, E)$ with a cost function $c_{\mathcal{I}} : E \rightarrow \mathbf{Q}^+$, and a non-negative penalty

PCST-ALG**Input:** An instance \mathcal{I} of rooted PCST**Output:** A tree T to span a subset of vertices (and deciding to pay the penalty for the rest)1. Run GW on instance \mathcal{I}_α with $c_{\mathcal{I}_\alpha} = c_{\mathcal{I}}$ and $\pi_{\mathcal{I}_\alpha} = \alpha\pi_{\mathcal{I}}$, for $\frac{1}{2} < \alpha < 1$.Let S be the set of vertices that are connected to the root in this solution.2. Run GW-U-GROWTH *without pruning* on instance \mathcal{I}_β with $c_{\mathcal{I}_\beta} = c_{\mathcal{I}}$ and, for $\beta > 1$, $\pi_{\mathcal{I}_\beta}(v) = \beta\pi_{\mathcal{I}}(v)$ for $v \in S$ and $\pi_{\mathcal{I}_\beta}(v) = 0$ otherwise.Let D be the set of vertices whose clusters even once get deactivated during GW-U-GROWTH.

3. Output the better of these two solutions:

First solution: GW tree on S from Step 1, denoted by $T^{GW}(S)$, and pay the penalties π of \overline{S} ; or**Second solution:** RZ tree on $S - D$, denoted by $T^{RZ}(S - D)$ and pay the penalties π of $\overline{S - D}$.Figure 1: A $(2 - \epsilon)$ -approximation algorithm for PCST.

function $\pi_{\mathcal{I}} : V \rightarrow \mathbf{Q}^+$. Our goal is to find a solution with minimum *overall cost*, i.e., construction cost plus penalty cost. The algorithm we describe will be for the rooted version of PCST, which means that the instance specifies some root node, which every feasible tree is required to include. Clearly, if we can solve this problem, then we can solve the unrooted version, by trying all roots and taking the best solution. Conversely, an algorithm for the unrooted version can be made to work for the rooted case simply by setting $\pi(\text{root}) = \infty$.

In the following proof, we work with small positive constants γ , δ , and ϵ in addition to constants α and β in the algorithm. The exact values of these parameters are determined in our main Theorem 21. We define $\pi(X)$, for $X \subseteq V$, to be $\sum_{x \in X} \pi(x)$. For ease of notation, in the rest of the proof we use OPT , c , and π , to mean $OPT_{\mathcal{I}}$, $c_{\mathcal{I}}$, and $\pi_{\mathcal{I}}$ (resp.), unless stated otherwise.

We start with the following theorem concerning the GW PCST algorithm.

Theorem 1 (Theorem 2.1 of [8], Theorem 4.1 of [13]). *The GW algorithm for PCST connects a set of vertices S via a tree T and pays the penalty for \overline{S} such that*

$$\sum_{e \in T} c(e) + 2\pi(\overline{S}) \leq 2OPT. \quad (2)$$

This result is stated as Theorem 2.1 of Chudak, et al. [8]. Theorem 4.1 of the original Goemans and Williamson paper [13] states a slightly weaker result, where the 2 on the LHS is replaced by a 1, but the stronger result is implicit in their proof.

We consider several cases in our proof. Case I and Case II are relatively easy to prove, but Case III is very involved and indeed is the heart of our analysis. Lemmas 4 and 5 are the main lemmas needed to deal with Case III.

Case I: Large penalty in GW. First, we start with the case in which $\pi(\overline{S}) \geq \gamma OPT$, i.e., in GW the penalty of \overline{S} is substantial compared to the optimum value. We use the ‘‘Lagrange-friendly’’ form of Theorem 1, i.e., the existence of the ‘‘2’’ on the left-hand side of inequality (7), which isn’t required if all one wants is a 2-approximation.

Lemma 2. *If $\pi(\overline{S}) \geq \gamma OPT$, then the first solution of PCST-ALG gives an approximation factor at most $2 - (2\alpha - 1)\gamma$.*

Proof. By Theorem 1, $c_{\mathcal{I}_\alpha}(T^{GW}(S)) + 2\pi_{\mathcal{I}_\alpha}(\bar{S}) \leq 2OPT_{\mathcal{I}_\alpha}$. Since we multiply penalties in \mathcal{I}_α by a factor $\alpha < 1$, $OPT_{\mathcal{I}_\alpha} \leq OPT$. Thus by definitions of $c_{\mathcal{I}_\alpha}$ and $\pi_{\mathcal{I}_\alpha}$, we have $c(T^{GW}(S)) + 2\alpha\pi(\bar{S}) \leq 2OPT$, which implies that $c(T^{GW}(S)) \leq 2OPT - 2\alpha\pi(\bar{S})$. The overall cost of the solution in PCST-ALG is $c(T^{GW}(S)) + \pi(\bar{S}) \leq 2OPT - 2\alpha\pi(\bar{S}) + \pi(\bar{S}) = 2OPT - (2\alpha - 1)\pi(\bar{S})$. By the assumption of the lemma, $\pi(\bar{S}) \geq \gamma OPT$ and by the fact that $2\alpha > 1$, $c(T^{GW}(S)) + \pi(\bar{S}) \leq 2OPT - (2\alpha - 1)\gamma OPT = (2 - (2\alpha - 1)\gamma)OPT$, as desired. \square

Case II: Large penalty in OPT. Now, we consider the case in which a substantial part of the optimum cost is the penalties.

Lemma 3. *Let O be the set of vertices connected to the root in OPT . If $\pi(\bar{O}) \geq \delta OPT$, then the first solution of PCST-ALG gives an approximation factor at most $2(1 - (1 - \alpha)\delta) = 2 - 2(1 - \alpha)\delta$.*

Proof. Again by Theorem 1, $c_{\mathcal{I}_\alpha}(T^{GW}(S)) + 2\pi_{\mathcal{I}_\alpha}(\bar{S}) \leq 2OPT_{\mathcal{I}_\alpha}$, which implies that $c(T^{GW}(S)) + 2\alpha\pi(\bar{S}) \leq 2OPT_{\mathcal{I}_\alpha}$ by the definitions of $c_{\mathcal{I}_\alpha}$ and $\pi_{\mathcal{I}_\alpha}$. The overall cost of the first solution of PCST-ALG is $c(T^{GW}(S)) + \pi(\bar{S}) \leq c(T^{GW}(S)) + 2\alpha\pi(\bar{S}) \leq 2OPT_{\mathcal{I}_\alpha}$, since $2\alpha > 1$. Thus, proving $OPT_{\mathcal{I}_\alpha} \leq (1 - (1 - \alpha)\delta)OPT$, i.e., $OPT - OPT_{\mathcal{I}_\alpha} \geq (1 - \alpha)\delta OPT$ proves the statement of the lemma. Since connecting O via the tree in OPT , namely $T^{OPT}(O)$, and paying the penalties of \bar{O} is a feasible solution for instance \mathcal{I}_α , $OPT_{\mathcal{I}_\alpha} \leq c_{\mathcal{I}_\alpha}(T^{OPT}(O)) + \pi_{\mathcal{I}_\alpha}(\bar{O})$. Thus proving a stronger statement $OPT - [c_{\mathcal{I}_\alpha}(T^{OPT}(O)) + \pi_{\mathcal{I}_\alpha}(\bar{O})] \geq (1 - \alpha)\delta OPT$ suffices. By the definitions of $c_{\mathcal{I}_\alpha}$ and $\pi_{\mathcal{I}_\alpha}$, the statement is equivalent to $\pi(\bar{O}) - \alpha\pi(\bar{O}) \geq (1 - \alpha)\delta OPT$ or $\pi(\bar{O}) \geq \delta OPT$, which is correct since it is the assumption of the lemma. \square

Case III: Wrong choice of terminals So far in both Cases I and II, we have only used the first solution obtained in PCST-ALG. Roughly speaking, when neither of conditions in Cases I and II happens, we are left with an instance in which the first solution in PCST-ALG performs poorly due to possible bad approximation factor of GW for Steiner tree or its wrong selection of vertices to connect. We handle the former case instance by using any Steiner Tree algorithm with a better guarantee than 2, say RZ, in this part. Now we consider the latter case. Since optimum pays a very small penalty (because we are not in case II), intuitively there are some vertices in S whose penalties are small and thus optimum pays their penalties, but we try to connect them to the root and thus incur a high overall cost. Of course, since we do not know the optimum solution, we can only try to find such vertices of small penalties ‘‘approximately.’’ Indeed, we show that set D in PCST-ALG can be this ‘‘approximate set’’ of vertices with small penalties. After removing D from S , we will solve the instance using any Steiner tree algorithm with a better guarantee than 2, such as RZ.

To give the reader an outline of the proof, we state the main lemmas here, and show how they combine to prove the main theorem. First we state that $\pi(D)$ can be made negligible by choosing a large enough β . This is fortunate since the algorithm will pay the penalties on $(S \cap D) \cup \bar{S}$.

Lemma 4. *For set D in PCST-ALG, $\pi(D) \leq 2OPT/\beta$.*

Next we state our main lemma, Lemma 5. Recall that the second solution within PCST runs a Steiner tree approximation on $S - D$ and pays the remaining vertices’ penalties. But how costly can it be to run the Steiner tree approximation on $S - D$? Here, we prove the key (nonalgorithmic) fact: Starting from the optimal tree spanning set O , the *marginal* cost of connecting to the vertices in $(S - D) - O$ is no larger than $2\beta\delta OPT$. (Newly introduced, δ can be made small enough that $2\beta\delta$ is arbitrarily small.) This guarantees existentially that there is a tree, not much more expensive than OPT , which spans all vertices in $S - D$; hence

a Steiner tree approximation algorithm (e.g., Robins-Zelikovsky) won't pay too much when fed terminal set $S - D$.

In Lemma 5, $S' = O \cap S$ and $A = (S - D) - O = S - D - S'$.

Lemma 5. *Suppose $\pi(\overline{O}) \leq \delta OPT$. Then there exists a forest F with the following properties:*

1. *For $A = (S - D) - O$, each node of A is included in one of the (non-empty) trees of F ;*
2. *each tree in the forest includes exactly one node from S' , and*
3. *$\text{cost}(F) \leq 2\beta\delta OPT$.*

Now it's just a matter of mopping up. Given the preparation, Lemma 6 is almost obvious. $OPT(1+2\beta\delta)$ is an upper bound on the cost of an optimal tree spanning $S - D$, so RZ will pay at most ρ times as much, if ρ is RZ's performance ratio; $\pi(\overline{S}) \leq \gamma OPT$, because we're not in Case I; and $\pi(D) \leq 2OPT/\beta$ by Lemma 4.

Lemma 6. *For case III, the second solution in PCST-ALG gives an approximation factor at most*

$$\rho(1 + 2\beta\delta) + \gamma + 2/\beta,$$

where $\rho \approx 1.55$ is the approximation factor of the RZ algorithm for Steiner tree.

Now we state (and prove) the final result:

Theorem 7. *Algorithm PCST is a $2 - \epsilon \leq 1.995$ approximation for PCST, for an $\epsilon \approx 0.005666$.*

Proof. By Lemmas 2, 3, and 6, since in PCST-ALG we take the better of the first and the second solutions, our overall approximation factor is at most

$$\max \{2 - (2\alpha - 1)\gamma, 2(1 - (1 - \alpha))\delta, [1.55(1 + 2\beta\delta) + \gamma + 2/\beta]\}.$$

One can just plug in $\alpha = 0.576717$, $\beta = 9.81908$, $\gamma = 0.036927$, $\delta = 0.00669276$, yielding an approximation factor of at most 1.994334142683, i.e., $2 - \epsilon$ with $\epsilon = 0.005665857317\dots$ or one can more eruditely reason that for $\alpha = 0.75$, one can choose β large enough and γ small enough that $\gamma + 2/\beta$ is negligible, and then realize that a suitably small positive δ will make all three terms less than 2. \square

In order to handle Case III, we will need to explore the guts of the GW algorithm, which we review in the next section. Readers who are already familiar with the GW algorithm will find nothing new in the next section, but we do introduce notation there that we will need later.

2.1 Inner workings of the GW PCST algorithm

Definition 8 (Clustering Forest). The first phase of GW (the moat growing, aka clustering, algorithm, GW-GROWTH) can be summarized in a forest \mathcal{F} of clusters. The leaves of the forest correspond of the singleton clusters the algorithm starts with. Every time two clusters C_1 and C_2 merge to form $C = C_1 \cup C_2$, we add a new node corresponding to C and make the nodes corresponding to C_1 and C_2 the children of this new node.

This is a “binary forest.” We now describe a way to color the edges of the graph. This gives us an alternate view of algorithm GW. To be more precise, it formalizes the intuition behind the algorithm. We do not simply color edges, but rather we color “edge portions.” In our view the graph is a geometric object, vertices being points and edges being *line segments* connecting their endpoints of length equal to their costs c_e . We will use length and cost interchangeably. Each portion of a line segment may be either uncolored or colored. The following defines how the colors are assigned.

Definition 9 (Coloring Scheme). The colors correspond to the nodes of \mathcal{F} . For each edge $e = (u, v)$, if the endpoints of e are in the same cluster, let w be their lowest common ancestor in \mathcal{F} . Let $p_u = (u_0 = \{u\}, u_1, \dots, u_k)$ be the path from u to w excluding w itself. Define p_v similarly. If the endpoints are in different clusters, let p_u and p_v be the respective paths from u and v to the root of their components (the paths include the root, unlike the previous case). Starting from u on the line segment corresponding to e , color the initial subsegment of length y_{u_0} with the color u_0 . Color the next subsegment of length y_{u_1} by color u_1 and so on. Do the same to the path p_v from the other side of the line segment corresponding to e .

First we should show that the coloring scheme is well-defined; i.e., we do not exhaust the length of the line segment before fully performing the coloring procedure. Note that

$$\sum_{q \in p_u} y_q + \sum_{q \in p_v} y_q = \sum_{\delta(C) \ni e} y_C \leq c_e.$$

The first equation follows from the structure of the laminar family and the inequality is due to the packing constraint present in the dual (how GW actually works). Thus, not only do we not run out of edge portion resources before our coloring is finished, but, in addition, we get the guarantee that an edge is *tight* if and only if it is fully colored.

The following lemma characterizes a nice property of this coloring.

Lemma 10. *For any C with $y_C > 0$, the subsegments of color C form a cut separating C from $V - C$. That is, any path connecting C and $V - C$ has a portion colored C . Furthermore, the length of this portion is at least y_C .*

Proof. The edges $\delta(C)$ are obviously a cut for paths connecting C to $V - C$. Each path from C to $V - C$ has to go through at least one of these edges. However, the line segment corresponding to each such edge has a portion colored C whose length is y_C (by definition of the coloring scheme). \square

Definition 11. For any cluster C , we define the *ball* of C , denoted by $B(C)$ to be the set of the subsegments having colors $C' \subseteq C$.

To avoid confusion, we refer to the components formed in the GW algorithm as *GW-clusters*. The algorithm GW works in two steps: GW-GROWTH and GW-PRUNE. The size of a GW-cluster C is defined to be $size(C) = \sum_{C' \subseteq C} y_{C'}$. For any GW-cluster C , define the potential of C at one point in time, $\phi(C)$, to be the difference between its aggregate penalty and its size: $\phi(C) = \pi(C) - size(C) = \sum_{v \in C} \pi_v - \sum_{C' \subseteq C} y_{C'}$. A GW-cluster C is *active* if and only if its dual variable y_C is growing. The growing is done at a rate of one per time unit. When two GW-clusters merge, the size of the new GW-cluster is the sum of the two old ones, and the same goes for the potential. A GW-cluster C is *inactive* in GW if and only if $size(C) = \pi(C)$ or $r \in C$. GW-GROWTH starts with a *clustering* of vertices into singleton GW-clusters. Each GW-cluster C not containing the root grows its dual variable y_C until one of the following happens:

- it merges with another components C' ; or

- it runs out of potential $\phi(C) = 0$.

The former happens if there is an edge e that has an endpoint C and the other in C' , such that $c_e = \sum_{C:e \in \delta(C)} y_C$. The algorithm maintains the following two conditions at all times: $\sum_{C:e \in \delta(C)} y_C \leq c_e \forall e$, and $\phi(C) \geq 0 \forall C$. As one sees in GW-GROWTH algorithm, the treatment of the root r and a GW-cluster containing it is different from normal vertices and GW-clusters. As first suggested by [15] we define an unrooted variant, GW-U-GROWTH, which unifies the treatment. Although this algorithm, followed by an appropriate pruning step, has been shown by [9] to give a 2-approximation for PCST, we do not use the solution directly. Instead, we simply use it to find the set D necessary for our algorithm. The set D is defined to be the set of vertices that are ever in an inactive GW-cluster during the run of GW-U-GROWTH. GW-U-GROWTH is the same as GW-GROWTH except that the GW-cluster containing the root always grows, until it either merges with the last other remaining active GW-cluster, or until the last remaining active GW-cluster deactivates.

2.2 Analysis

Lemma 12. *Let T be a tree spanning all the vertices in S . Further, let $C \subseteq S$ be an inactive GW-cluster (at some time τ during the run of GW-U-GROWTH), which is a proper subset of S . Then, the intersection of $B(C)$ and the line segments corresponding to T has cost at least $\pi(C)$.*

Proof. Look at all the GW-clusters $C' \subseteq C$. Clearly the subsegments corresponding to different colors are disjoint. Since each color specifies a cut, by Lemma 10, the intersection with each color C' is at least $y_{C'}$. Thus the total intersection with $B(C)$ is at least $\sum_{C' \subseteq C} y_{C'} = \text{size}(C) = \pi(C)$, where the last equation follows from C being inactive in GW-U-GROWTH. \square

If S is the subset of vertices connected at the end ($r \in S$), then all vertices in S end up being in an inactive GW-cluster S . A GW-cluster not containing the root r is called *normal*. However, D is the set of vertices that are once part of an inactive normal GW-cluster. Thus, all vertices of D have a final reactivation.

Lemma 13. *Consider the last inactive GW-cluster any vertex $v \in D$ was in. The balls of distinct such GW-clusters cannot share any subsegment.*

Proof. Each vertex only appears in one such GW-cluster, namely in the (inactive) GW-cluster that contained it right before its final reactivation. Assume for the sake of contradiction, that a subsegment appears in two such GW-clusters. The subsegment is colored with GW-cluster, say C . If two balls share this edge, both should contain C . Let them be $C \subseteq C_1 \subseteq C_2$, where $r \notin C_2$. We run into a contradiction, as C_1 cannot be the last inactive normal GW-cluster for any vertex. Any vertex in C_1 is also included in C_2 which is inactive at a later time than C_1 is and is also normal. \square

Now we prove Lemma 4, which was stated earlier.

Proof of Lemma 4: Look at vertices in D when they were last inactive. Let them be in GW-clusters C_1, C_2, \dots, C_k at the time of their last deactivation. The balls corresponding to these GW-clusters have to be disjoint by Lemma 13. The tree T^{GW} connects up all the vertices in S . By Lemma 12, the intersection of the tree with each ball is at least as much as the total penalty of the ball (which is equal to its size). Thus the cost of the tree is at least β times the total penalty of vertices in D . As the cost of the tree is less than $2OPT$, this means that $\Pi(D) \leq 2 \cdot OPT/\beta$. \blacksquare

Let $S' = O \cap S$ be the intersection of the set of nodes spanned by OPT and those spanned by GW in the first step of PCST-ALG. Let $A = S - S' - D$. Lemma 5, state earlier, shows that there exists an

inexpensive forest F of edges connecting A to S' . The notation A was chosen as a mnemonic for *augment*, because even if we *augment* O by A , we can still span $O \cup A$ cheaply, paying OPT to span O , plus $\text{cost}(F)$ to augment the optimal tree to additionally span A . Therefore, if we run a good approximation algorithm for Steiner tree, such as RZ, on $O \cup A$ or any subset thereof, it should yield an inexpensive tree.

To make this argument, it is critical that we are solving the *rooted* PCST problem (not the unrooted version), because both S and O must contain the root node, which guarantees that $S' \neq \emptyset$. We now define a set of GW-clusters that play an important role in this lemma.

Definition 14. Let $\mathcal{G} = \{\text{GW-clusters } C : C \subseteq \overline{S'}\}$.

Since $\overline{S'} \subseteq \overline{O} \cup \overline{S}$ and $\pi_{\mathcal{I}_\beta}(\overline{S}) = 0$, we have that $\sum_{C \in \mathcal{G}} y_C \leq \pi_{\mathcal{I}_\beta}(\overline{O}) \leq \beta\pi(\overline{O})$, where the y_C are the dual variables generated by running GW-U-GROWTH on \mathcal{I}_β . If we find ourselves in Case III, it means that $\pi(\overline{O})$ is small, and hence so is $\sum_{C \in \mathcal{G}} y_C$. Thus, if we can upper bound $\text{cost} F$ in terms of $\sum_{C \in \mathcal{G}} y_C$, we will be in good shape.

The proof of Lemma 5 is more complicated than those of the previous ones. The vertices in A never got deactivated during GW-U-GROWTH before being connected to the root. Intuitively, this implies that vertices in A are close to the set S' , compared to their penalties.¹ Hence, we can charge the cost of their connection against their penalties.

In Lemma 5, stated earlier, which we now prove, a forest F consists of an acyclic set of edges, and an associated set V_F of vertices incident to them. In other words, F is a collection of disjoint trees. Isolated vertices (i.e., empty trees) are not considered to be part of F .

Proof of Lemma 5: If $A = \emptyset$, we can take $F = \emptyset$ and the result is trivial. Otherwise, let T be the tree that is generated by GW-U-GROWTH in instance \mathcal{I}_β . We will derive F from T by deleting some edges in two phases. Recall that GW-U-GROWTH finds a tree that spans the entire vertex set V (and it does not include a pruning phase).

Thus, T already satisfies property 1, and both phases of edge deletion will preserve it as an invariant. The first phase will accomplish property 2, and the second will preserve it as an invariant. Both phases of edge deletion taken together will ensure that

$$\text{cost}(F) \leq 2 \sum_{C \in \mathcal{G}} y_C, \quad (3)$$

where the y_C are the dual variables generated by GW-U-GROWTH, and \mathcal{G} is the set of GW-clusters that are entirely contained in $\overline{S'}$. But this sum is at most the total prize of $\overline{S'}$ in \mathcal{I}_β . Since \overline{S} gets no prize in \mathcal{I}_β , this is $\beta\pi(S - S') \leq \beta\pi(\overline{O}) \leq \beta\delta OPT$, where the inequalities follow from $S - S' \subseteq \overline{O}$ and the lemma hypothesis. This sequence of inequalities will imply property 3.

We note that our description of F is constructive in the mathematical sense but not the computational sense, because it will depend on the set $S' = O \cap S$, and our algorithm does not know O . We now show how to construct F by deleting edges from T . To help us prove property 3, we will also ensure that F satisfies the following additional property.

If u, v are distinct vertices in S' , let P_{uv} denote the path from u to v in T , and let e_{uv} be the last edge on P_{uv} that was added to T in GW-U-GROWTH. For each such pair u, v , $e_{uv} \notin F$. (4)

Start with $F = T$. As we observed above, T spans all of V . Consider the edges of T in the reverse of the order in which they were added during GW-U-GROWTH. When we arrive at an edge e , if there exist

¹We say close to S' , rather than close to the root, because, unlike A , the nodes in S' could have significant penalties, so that once a GW-cluster $C \in \mathcal{G}$ merges with one, it could remain active for quite a while longer.

two vertices in S' that are still in the same GW-cluster of F but would be separated by deleting e , then do so. These deletions preserve the invariant that each tree in the forest F contains at least one node from S' . This invariant ensures that each node of A will always be in the same tree of F as some node from S' ; in particular, it will be incident to some edge in F , so property 1 is preserved. We will satisfy property 4, because at the time we consider edge e_{uv} , nodes u and v are still connected by the path P_{uv} , so we delete e_{uv} . Thus, F will end up with exactly one node of S' per tree, satisfying 2. We need to do further pruning in order to achieve property 3.

Next, we will prune each tree in the forest F in a very similar way to what the pruning phase of GW would do. The pruning rule will be based on some special GW-clusters.

Definition 15. If GW-cluster $C \subseteq \overline{S'}$ and C became inactive at any point during GW-U-GROWTH, we say that C is *colored gray*.

Pruning rule: If there exists a gray GW-cluster whose intersection with one of the trees in F is precisely the entire subtree to one side of some edge e , then prune away e and this entire subtree. (5)

We apply the pruning rule iteratively, until there are no longer any gray GW-clusters to which it applies. Note that the gray GW-clusters contain (by definition) only nodes from $D - S'$, so this pruning does not delete any node from A or S' , since both are disjoint from $D - S'$. Moreover, since we prune only entire subtrees, we do not break apart any trees into pieces. Thus, properties 1 and 2 are maintained. The remaining forest is our final F .

We now prove that this F satisfies (3). The proof is very similar to the proof of the approximation guarantees in Goemans-Williamson. First let us refine the coloring scheme of Definition 9. GW-U-GROWTH has certain event points (an edge going tight or an active GW-cluster becoming inactive) that unfold over time. Let $0 = t_0 \leq t_1 \leq t_2 \leq \dots$, where t_j is the time of the j^{th} event point, and define the j^{th} epoch to be the interval of time from t_{j-1} to t_j . The coloring scheme of Definition 9 partitions the length of each tight edge e from GW-U-GROWTH amongst the GW-clusters C that “pay for” the edge. For the portion of edge e colored by GW-cluster C , let us further subdivide this length into epochs, where epoch j gets the amount by which GW-cluster C grew during epoch j , which is $t_j - t_{j-1}$ if it was active during this epoch, and 0 otherwise. Similarly, the dual variables y_C can be partitioned into epochs, where $y_C^{(j)}$ is the amount by which y_C grew during epoch j . We will now prove that the total length of edge segments from F that are labeled by epoch j is at most $2 \sum_{C \in \mathcal{G}} y_C^{(j)}$. Summing over the epochs yields (3).

We now analyze an arbitrary epoch j . Let \mathcal{C}_j denote the set of GW-clusters that existed during epoch j and contain a node from V_F . The entire set of GW-clusters that were alive during epoch j partitions V , but we will be concerned only with the projection of this partition onto V_F , and hence we are interested only in the GW-clusters in \mathcal{C}_j . Consider the graph (V_F, F) , then collapse each GW-cluster $C \in \mathcal{C}_j$ into a supernode. Call the resulting graph H . It may be the case that some GW-cluster $C \in \mathcal{C}_j$ contains nodes from distinct trees in the forest F , in which case H will have fewer distinct connected components than F did. Although the nodes of H are identified with GW-clusters in \mathcal{C}_j , we will continue to refer to them as GW-clusters, in order to avoid confusion with the nodes of the original graph. Some of the GW-clusters are active and some of may be inactive. Let us denote the active and inactive GW-clusters in \mathcal{C}_j by \mathcal{C}_A and \mathcal{C}_I , respectively. The edges of F that include a non-zero length labeled by epoch j are exactly those edges of H that are incident to an active GW-cluster, and the total amount of these edges that is paid off during epoch j is $(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_A} \deg_H(C)$. Since every active GW-cluster in \mathcal{G} grows by exactly $t_j - t_{j-1}$ in epoch j , we have $\sum_{C \in \mathcal{G}} y_C^{(j)} \geq \sum_{C \in \mathcal{G} \cap \mathcal{C}_j} y_C^{(j)} = (t_j - t_{j-1}) |\mathcal{G} \cap \mathcal{C}_A|$. Thus, it suffices to show that $\sum_{C \in \mathcal{C}_A} \deg_H(C) \leq 2 |\mathcal{G} \cap \mathcal{C}_A|$.

Let us call the GW-clusters in $\mathcal{G} \cap \mathcal{C}_A$ *hungry* and the active or inactive GW-clusters that intersect S' *sated*. Thus, we want to show that the sum of the degrees of the active GW-clusters in H is at most twice the number of hungry GW-clusters. First we must make some simple observations about H . Since T is a tree, $E(F) \subseteq E(T)$, and each GW-cluster represents a disjoint induced subtree of T , the contraction to H introduces no cycles. Thus, H is a forest. Every inactive leaf of H must be sated, because otherwise the corresponding GW-cluster C would be gray and satisfy the conditions of the pruning rule (5), and it would have been pruned away. Finally, within each tree of the forest H , there is exactly one sated GW-cluster. This may seem obvious because each tree of F contains exactly one node of S' , but it can be the case that some $C \in \mathcal{C}_j$ contains nodes from multiple trees of F , so when we contract C , these multiple trees become a single tree in H . We will use property 4 to show that when this happens, the nodes in S' from the distinct trees of F that were merged are actually in the same sated GW-cluster during this epoch.

Suppose on the contrary that there exist two distinct (and hence disjoint) sated GW-clusters in the same tree of H . Each of these GW-clusters contains a vertex from S' (call them u and v), and consider the path P_{uv} from u to v in T . During epoch j in GW-U-GROWTH, the only edges on this path that had not yet been added to T are those that appear on the path P'_{uv} in H between these two sated GW-clusters. Since edge e_{uv} is the last edge on path P_{uv} to be added to T , it must be in P'_{uv} . But this is a contradiction, because our first pruning stage removed e_{uv} from F .

Here is another way of looking at this argument. If u, v are distinct nodes of S' , they are in different trees of F (by property 2), but they could both end up getting collapsed into the same tree of H . But this can happen only if some GW-cluster C exists in epoch j that contains nodes from both u 's tree and v 's tree in F . But this can be the case only if epoch j occurs after the edge e_{uv} has been added to the solution being built up by GW-U-GROWTH, at which point all of P_{uv} has been added, so C contains both u and v .

With this information about H , it is easy to bound $\sum_{C \in \mathcal{C}_A} \deg_H(C)$. Let k be the number of trees in H , and l be the number of inactive sated leaves in H . There exist exactly k sated GW-clusters in \mathcal{C}_j (one per tree of H), so there are at most $k - l$ active sated ones, since at least l are inactive. Since every active cluster is either sated or hungry, the set of hungry ones, $\mathcal{G} \cap \mathcal{C}_A$ satisfies

$$|\mathcal{G} \cap \mathcal{C}_A| \geq |\mathcal{C}_A| - (k - l). \quad (6)$$

The number of GW-clusters in H is $|\mathcal{C}_A| + |\mathcal{C}_I|$, so the number of edges is $|\mathcal{C}_A| + |\mathcal{C}_I| - k$, hence $\sum_{C \in \mathcal{C}_A \cup \mathcal{C}_I} \deg_H(C) = 2(|\mathcal{C}_A| + |\mathcal{C}_I| - k)$. But the sum of the degrees of the inactive GW-clusters is at least $2|\mathcal{C}_I| - l$, since the only inactive leaves are the l sated ones. Thus, $\sum_{C \in \mathcal{C}_A} \deg_H(C) \leq 2(|\mathcal{C}_A| + |\mathcal{C}_I| - k) - (2|\mathcal{C}_I| - l) = 2|\mathcal{C}_A| - k + l \leq 2|\mathcal{G} \cap \mathcal{C}_A|$, by (6) as we wished to show. ■

3 Prize-collecting TSP

It is standard for prize-collecting TSP to assume that the triangle inequality holds (see e.g., [13]). Our algorithm for prize-collecting TSP, called PCTSP-ALG, is exactly the same as PCST-ALG with these differences that we run the PCTSP algorithm of Goemans and Williamson [13] (Section 4.3.2) instead of the GW algorithm for PCST and we run CHRISTOFIDES algorithm, the well-known $\frac{3}{2}$ -approximation for metric TSP, instead of RZ. We have the following theorem similar to Theorem 1.

Theorem 16 (Section 4.3.2 of [13]). *GW algorithm for TSP connects a set of vertices S via a tour $T(S)$ and pays the penalty for \bar{S} such that*

$$\sum_{e \in T(S)} c(e) + 2\pi(\bar{S}) \leq 2OPT. \quad (7)$$

The analysis for the cases in which we have large penalty in GW or we have large penalty in OPT is identical to that of cases I or case II for PCST-ALG. Thus here we have also similar lemmas to Lemma 2 and Lemma 3.

Lemma 17. *If $\pi(\overline{S}) \geq \gamma OPT$, then the first solution of PCTSP-ALG gives an approximation factor at most $2 - (2\alpha - 1)\gamma$.*

Lemma 18. *Let O be the set of vertices in the tour of OPT . If $\pi(\overline{O}) \geq \delta OPT$, then the first solution of PCTSP-ALG gives an approximation factor at most $2(1 - (1 - \alpha)\delta) = 2 - 2(1 - \alpha)\delta$.*

Indeed the analysis for the third case, i.e., when we have the wrong choice of terminals, is also very similar to the case III in the analysis for PCST-ALG. Here we mention the differences. Before this first we need to mention how GW algorithm for TSP works. Indeed it just applies GW algorithm for PCST on the instance with $\pi' = \frac{1}{2}\pi$, pays the penalties of vertices not in the resulting Steiner tree, duplicates the edges of the resulting Steiner tree and shortcuts it to obtain the desired tour. In our algorithm, when we run GW-U-GROWTH *without pruning* on the instance \mathcal{I}_β in our PCTSP-ALG to obtain the set D , indeed our β has a coefficient $\frac{1}{2}$ in it. Now, knowing the exact GW algorithm for PCTSP, it is easy to see that we have the following lemma, whose proof is the same as that of Lemma 4, except that the intersection of the optimal solution with each ball is twice the size of the ball.

Lemma 19. *For set D in PCST-ALG, $\pi(D) \leq OPT/\beta$.*

Finally to show that $S - D$ has a good TSP, we build one. Start with a TSP for $S' = S \cap O$, which is not costlier than OPT because of the triangle inequality. As we saw in Lemma 5, there is a forest to connect $S - S' - D$ to S' . By doubling the edges in each component and finding an Eulerian tour, we end up with a cycle (the TSP of S') with some ears. By shortcutting, we get a TSP for $S - D$ whose cost is no more than $OPT(1 + 4\beta\delta)$. Thus we conclude with the following theorem whose proof is identical to that of Lemma 6

Lemma 20. *For case III, the second solution in PCTSP-ALG gives an approximation factor*

$$\psi(1 + 4\beta\delta) + \gamma + 1/\beta$$

where $\psi = 1.5$ is the current best approximation factor for TSP (the approximation factor of CHRISTOFIDES algorithm).

Now by combining Lemmas 17, 18, 20, and picking $\alpha = 0.58$, $\beta = 4.45435$, $d = 0.0084$ and $\gamma = 0.0439$, we obtain the main theorem of this section.

Theorem 21. *Algorithm PCTSP-ALG is a $2 - \epsilon \approx 1.992976$ approximation for PCST, for an $\epsilon \approx 0.007024$.*

References

- [1] Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.
- [2] Aaron Archer, Asaf Levin, and David P. Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM J. Comput.*, 37(5):1472–1498, 2008.
- [3] Sanjeev Arora and George Karakostas. A $2 + \epsilon$ approximation algorithm for the k -MST problem. *Mathematical Programming*, 107(3):491–504, 2006.

- [4] Sunil Arya and Hariharan Ramesh. A 2.5 factor approximation algorithm for the k -MST problem. *Information Processing Letters*, 65(3):117–118, 1998.
- [5] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.
- [6] Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David P. Williamson. A note on the prize collecting traveling salesman problem. *Math. Program.*, 59:413–420, 1993.
- [7] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, pages 36–45, 2003.
- [8] Fabian A. Chudak, Tim Roughgarden, and David P. Williamson. Approximate k -MSTs and k -Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming*, 100:411–421, 2004.
- [9] Paulo Feofiloff, Cristina G. Fernandes, Carlos Eduardo Ferreira, and José Coelho de Pina. A note on johnson, minkoff and phillips’ algorithm for the prize-collecting steiner tree problem. Manuscript available from <http://www.ime.usp.br/cris/publ/jmp-analysis.ps.gz>, 2006.
- [10] Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 302–309, 1996.
- [11] Naveen Garg. 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*, pages 396–402, 2005.
- [12] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–316, 1992.
- [13] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.
- [14] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.
- [15] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: theory and practice. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.
- [16] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.