Improved Approximation Algorithms for PRIZE-COLLECTING STEINER TREE and TSP

Aaron Archer* MohammadHossein Bateni† MohammadTaghi Hajiaghayi* Howard Karloff*
* *AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932.*
E-mail: {aarcher,hajiagha,howard}@research.att.com.
† *Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540.*
E-mail: mbateni@cs.princeton.edu.

Abstract— We study the prize-collecting versions of the Steiner tree, traveling salesman, and stroll (a.k.a. PATH-TSP) problems (PCST, PCTSP, and PCS, respectively): given a graph (V, E) with costs on each edge and a penalty (a.k.a. prize) on each node, the goal is to find a tree (for PCST), cycle (for PCTSP), or stroll (for PCS) that minimizes the sum of the edge costs in the tree/cycle/stroll 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 by AT&T to the optimization of real-world telecommunications networks. The most recent improvements for the first two problems, giving a 2-approximation algorithm for each, appeared first in 1992. (A 2-approximation for PCS appeared in 2003.) The natural linear programming (LP) relaxation of PCST has an integrality gap of 2, which has been a barrier to further improvements for this problem.

We present $(2 - \epsilon)$-approximation algorithms for all three problems, connected by a unified technique for improving prize-collecting algorithms that allows us to circumvent the integrality gap barrier.

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*) π_v for each node $v \in V$. In PCST, we must select some tree T spanning a set $V(T)$ of nodes so as to minimize the combined cost

$$\sum_{e \in T} c_e + \sum_{v \in \overline{V(T)}} \pi_v = c(T) + \pi(\overline{V(T)}),$$

where $\overline{V(T)} = V - V(T)$, and the notation $c(T)$ and $\pi(\overline{V(T)})$ follows the usual convention that when x is a vector and S is a subset of its indices, $x(S) = \sum_{i \in S} x_i$. That is, we aim to minimize the total cost of the tree plus the penalties of the nodes not spanned. Although T denotes a set of *edges*, in a slight abuse of notation we will often write $\overline{T}$ to denote the set $\overline{V(T)}$ of *nodes*. Since we will never take the complement of an edge set, this should cause no ambiguity. PCTSP is the same, except that the set of edges should form a cycle instead of a tree. We also study a variant of PCTSP, the *prize-collecting stroll* (PCS) problem, in which the edges should form a stroll (i.e., path). When $\pi_v = \infty$ for all $v \in V$, PCTSP simplifies to the ordinary TSP because we have no choice but to visit all nodes, and PCS simplifies to PATH-TSP, the path version of TSP. Similarly, if each penalty is 0 or ∞, PCST simplifies to an instance of the ordinary Steiner tree problem; in this case, the nodes $\{v : \pi_v = \infty\}$ are called *terminals* (which the Steiner tree *must* connect) and the nodes $\{v : \pi_v = 0\}$ are called *Steiner* nodes (which the tree may use if it helps). These are all notorious NP-hard problems, so PCST, PCTSP, and PCS are NP-hard as well. Thus, we are interested in obtaining approximation algorithms for them.

In PCTSP and PCS, one typically assumes that the graph is complete and the edge costs satisfy the triangle inequality. Otherwise the problem is inapproximable, since determining whether the edges of cost zero form a Hamiltonian cycle or Hamiltonian path is NP-hard. These assumptions open the door to algorithms that use "shortcutting." In PCST, these extra assumptions have no effect.

In the *rooted* version of these problems, there is a specified root node r that must be spanned. Since the rooted and unrooted versions are reducible to each other,[1] we consider only the rooted version for the rest of the paper.

The first approximation algorithms for the PCST and PCTSP problems were given by Bienstock et al. [7], 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 [18] constructed primal-dual algorithms using the same LP relaxation to obtain a 2-approximation for both problems, building on work of Agrawal, Klein and

[1]To use an algorithm for the unrooted case to solve the rooted case, just set $\pi_r = \infty$. To go the other direction, just try all possible roots.

Ravi [1]. Chaudhuri et al. modified the Goemans-Williamson algorithm to achieve a 2-approximation for PCS [8]. In a 1998 conference talk covering unpublished work [15] , Goemans mentioned (exactly) two open problems: resolving whether the integrality gap of the Held-Karp LP relaxation for TSP is exactly $\frac{4}{3}$ and improving the PCTSP approximation ratio below 2. The conference version of the Goemans-Williamson paper [17] appeared in 1992; in the 17 years since, there have been no improved approximation algorithms invented for either PCST or PCTSP, until now. This paper gives $(2-\epsilon)$-approximation algorithms for both problems, and for PCS as well.

By contrast, improvements for the original Steiner tree problem came fast and furiously after Zelikovsky's announcement, in 1990, of an $11/6$-approximation for Steiner tree ([28], [29]), the first algorithm to beat the naive 2-approximation. Berman and Ramaiyer's improvement of the $11/6 = 1.833...$ bound of Zelikovsky to 1.746 appeared in 1992 [6]. Zelikovsky himself improved Berman and Ramaiyer's 1.746 to 1.693 in 1996 [30], which was followed by Prömel and Steger's improvement to 1.667 in 1997 [26], and by Karpinski and Zelikovsky to 1.644 in 1997 [25], after which Hougardy and Prömel improved the bound to 1.598 in 1999 [21]. The best and most recent bound, just under 1.55, is due to Robins and Zelikovsky, in 2005 [27].

We hope for our work to trigger similar improvements for PCST, PCTSP, and PCS. Indeed, this process has already begun. After we informed Goemans of our new result, he combined some of our ideas with others from [15] (namely, a randomized version of the PCTSP algorithm from [7]) to improve the ratio for PCTSP below 1.915 [16].

Although the value of ϵ that we achieve for the two prize-collecting problems is small (about 0.01 in all three cases), this is a conceptual breakthrough, since the factor 2 was thought to be a barrier. The natural LP relaxation for PCST used in [7] and [18] is known to have an integrality gap of 2. Thus, it cannot by itself provide a strong enough lower bound 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. Moreover, work on the PCST has had a large impact, both in theory and practice. At AT&T, PCST code has been used in large-scale studies in access network design, both as described in Johnson, Minkoff and Phillips [23], and in unpublished applied work involving the first author of the present paper.

The impact of PCST within approximation algorithms is also far-reaching. As noted by Chudak, Roughgarden and Williamson [10], 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 give a formal definition shortly). This implies the 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 pays at most $\pi(\overline{V(T)})$ penalty. Thus, if one runs PCST with all penalties set to some constant, and it happens to return a tree spanning exactly k nodes, then this tree is a 2-approximate k-MST. This property has been used in a sequence of papers ([13], [4], [3]) culminating in a 2-approximation algorithm for k-MST by Garg [14]. It has also been used to improve the approximation ratio and running time of algorithms for the MINIMUM LATENCY problem ([2], [8]). The technique of applying Lagrangian relaxation to a problem with hard constraints, then 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 [22].

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

$$c(T) + 2\pi(\overline{T}) \leq 2OPT. \tag{1}$$

To be an ordinary 2-approximation, it would suffice to satisfy (1) with the 2 on the left side changed to 1. One way to view this is that GW essentially achieves an approximation ratio of 1 on the penalty term, and 2 on the tree term. We can show that if either GW or OPT pays a large amount of penalty, then GW achieves a $(2-\epsilon)$-approximation. In the case where both GW and OPT pay a small penalty, intuitively GW is finding approximately the correct set of nodes to span, but it might have failed to construct a cheap tree. (This intuition is somewhat misleading; in actuality, GW might choose a very wrong set of nodes, so we will have to correct for this as well.) 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 for the ordinary Steiner tree problem, using those nodes as terminals. We use RZ here as a black box: for the purpose of achieving a $(2-\epsilon)$ factor for PCST, any $(2-\epsilon')$-approximation for ordinary Steiner tree would suffice. We emphasize that in selecting the terminal set to send to RZ, we do *not* simply use the set of nodes that GW chose to span; in fact, we carefully select a subset. The proper selection of these terminals is the crux of our algorithm.

There are two keys to this whole scheme. First, we must ensure that the extra nodes that GW spanned but that we choose not to designate as terminals have a small total penalty. We do this by using LP duality to bound their penalty against the cost of the tree produced by GW (Lemma 3). The second key is Lemma 4, which guarantees that there exists a tree spanning our chosen terminals 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 (since both the nodes excluded by GW and the extra ones that we excluded from the terminal set have small total penalty). Our algorithm for PCTSP is very similar, except that we use the Christofides $\frac{3}{2}$-approximation algorithm for TSP [9] instead of RZ. For PCS, we use a $\frac{3}{2}$-approximation for the path version of TSP, due to Hoogeveen [20].

One novel feature of our analysis is that we use the details of the GW moat-growing algorithm to prove in Lemma 4 that there *exists* an inexpensive set of edges satisfying certain properties, but not to find it. All other applications of GW of which we are aware use it purely in an algorithmic sense to identify edges; this may be the first time it has been utilized solely to provide an existence proof.

We structure the rest of the paper as follows. Section 2 describes the GW PCST algorithm. Section 2.1 discusses performance guarantees for GW, along with a bad instance that motivates our improved algorithm. Section 3 describes our algorithm for PCST, states the key lemmas, and fits them together to prove the approximation ratio of $(2-\epsilon)$. The heart of our analysis appears in Section 3.1, where we prove the key lemmas. Section 4 applies the same techniques to derive $(2-\epsilon)$-approximation algorithms for PCTSP and PCS. Section 5 explains how several variants of GW relate.

2. Goemans-Williamson algorithm

This section describes the GW algorithm [18] and some of the reasons why it is difficult to modify it to improve its approximation ratio. The algorithm we describe below is a slight variant of that given in [18], as suggested by Johnson, Minkoff and Phillips [23].

Let $\mathcal{I}$ be our input instance, in which we are given a graph $G=(V,E)$, a root node $r \in V$, a cost c_e for each edge e, and a penalty π_v for each node v. Recall that our goal is to find a tree T, rooted at r, that minimizes $c(T)+\pi(\overline{T})$. Since every feasible tree is required to include r, without loss of generality we can set $\pi_r=\infty$. The GW algorithm is motivated by the following LP relaxation and its dual.

$$\begin{aligned}
&\text{Primal: minimize } \sum_{e\in E} c_e x_e + \sum_{R\subseteq V-\{r\}} \pi(R) z_R\\
&\text{subject to: } \sum_{e\in\delta(R)} x_e + \sum_{U\supseteq R} z_U \geq 1 \quad \forall R\subseteq V-\{r\}\\
&x_e, z_R \geq 0 \quad \forall e\in E, R\subseteq V-\{r\}
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\text{Dual: maximize} && \sum_{R\subseteq V-\{r\}} y_R\\
&\text{subject to:} && \sum_{R:e\in\delta(R)} y_R \leq c_e && \forall e\in E\\
&\sum_{U\subseteq R} y_U \leq \pi(R) && && \forall R\subseteq V-\{r\}\\
&y_R \geq 0 && && \forall R\subseteq V-\{r\}.
\end{aligned} \tag{3}$$

Here, $\delta(R)$ denotes the set of edges with exactly one endpoint in R. A tree T rooted at r corresponds to the following solution in the primal LP: $x_e=1$ for each $e\in T$, $z_{\overline{T}}=1$, and all other variables are zero. In light of this, the primal constraint for set R says that either $R\subseteq\overline{T}$, or else T contains an edge in $\delta(R)$. We will refer to the first set of dual constraints as *edge packing* constraints, and the second set as *penalty packing* constraints.

The algorithm consists of two phases, GW-Growth then GW-Pruning. In GW-Growth, the algorithm maintains a forest F of edges between nodes in V. We refer to the connected components of F as *clusters* (or GW-clusters). Initially, $F=\emptyset$, so the clusters are all singletons. We identify a cluster with the set of nodes it spans. Associated with each cluster C there is a dual variable y_C from LP (3). For the purpose of this description, we also assign a "fake" dual variable y_C to the cluster that contains r, even though this variable does not appear in LP (3).

The algorithm has a notion of time, which unfolds in *epochs*. An epoch ends when an *event point* is reached, at which time the event is processed and the next epoch begins. If two events are set to occur simultaneously, then ties can be broken arbitrarily, although processing the first event may prevent the second from occurring. If both events do occur, then the epoch between them has zero duration. There are two types of event points, *tight edge* and *cluster death* events, which we describe momentarily. Initially all y_C's are 0. During each epoch, each cluster is either *active* or *dead* (a.k.a. *inactive*), and each active cluster C increases its y_C value at rate 1 for the duration of the epoch. At time 0, all clusters are active. Each cluster C also has a *potential* P_C. Initially, $P_{\{v\}}=\pi_v$ for all $v\in V$. During each epoch, the potential of each active cluster decreases at rate 1, in order to maintain the invariant

$$P_C = \pi(C) - \sum_{U\subseteq C} y_U. \tag{4}$$

If the potential of a cluster ever reaches zero, it triggers a cluster death event, wherein the cluster dies (i.e., becomes inactive). Since $P_{\{r\}}=\pi_r=\infty$, the root cluster never dies. Note that a cluster has zero potential if and only if its penalty packing constraint is tight.

A tight edge event is triggered on e when its edge packing constraint becomes tight (where we include in the sum the "fake" dual variables corresponding to the root cluster). When such an e goes tight, we add e to F, causing the two components C_1 and C_2 to merge. We set the new potential $P_{C_1\cup C_2}=P_{C_1}+P_{C_2}$, to maintain invariant (4), and declare $C_1\cup C_2$ to be active. Thus, a dead cluster can be resurrected if an active cluster merges with it.

The purpose of the two types of events is to prevent the penalty and edge packing constraints from being violated. If processing one event obviates the need for another event that would otherwise have been triggered at the same time, we skip the latter one. Thus, we trigger the death of cluster C only if C is still an active cluster. Similarly, we trigger a tight edge event on e only if e connects two distinct clusters C_1 and C_2 and at least one of them is active. Because of this rule, F remains a forest. If the edge packing constraints for two distinct edges between C_1 and C_2 become tight simultaneously, then processing one of them will merge the components, and this rule prevents the other event from occurring. Similarly, this rule can cause tight edge and cluster death events to prevent each other from occurring.

Since the root cluster is always active, it will eventually merge with every other cluster. At this point, F is a tree spanning all of V, and the GW-GROWTH phase ends. The algorithm works even with infinite penalties.

A good intuitive way to understand GW-GROWTH is that each dual variable y_C represents a *moat*. As the moat grows, it "fills up" the edges at the boundary of cluster C.[2] Whenever the moats have collectively filled up an edge, we "buy" that edge by putting it in F. Note that every cluster C is a contiguous set of nodes in the final forest F.

The set of edges we have bought may be too expensive, so we must now prune some of them away, in the GW-PRUNING phase. If a cluster *ever* died during the course of GW-GROWTH, color it black. If there exists a black cluster C such that the intersection of C with $V(F)$ consists of the vertex set of the entire subtree of F to one side of some edge e, then prune away e and this entire subtree. Clearly, F remains a tree. We iteratively prune away subtrees of F in this fashion, in any order, until there are no more such black clusters. At this point, GW-PRUNING ends and the algorithm returns the tree F.

2.1. A good case, and a motivating bad example

Our starting point is the following theorem regarding GW.

Theorem 1 ([18], [10], [11])**.** *The tree T returned by the* GW *algorithm for* PCST *satisfies*

$$c(T) + 2\pi(\overline{T}) \leq 2OPT. \tag{5}$$

The history of Theorem 1 is somewhat complicated; see Section 5. Notice that (5) is stronger than what would be necessary for GW to be a 2-approximation: there is a factor of 2 in front of $\pi(\overline{T})$, where a 1 would suffice. Because of this property, GW is said to be a *Lagrangian-preserving* 2-approximation algorithm.[3] This stronger guarantee obviously implies that if $\pi(\overline{T})$ is large compared to OPT, then T is better than a 2-approximation. The following theorem quantifies this, and shows we can also win if a significant fraction of OPT comes from the penalties.

[2]This "geometric" view of the dual variables as moats was suggested by Jünger and Pulleyblank [24], predating GW.

[3]The term arises because of its connections to Lagrangian relaxation, which we will not explore in this work.

Lemma 2. *Let O denote the optimal tree, and let $\delta = \frac{\pi(\overline{O})}{OPT}$. Fix any $\alpha \in [\frac{1}{2}, 1]$, and derive a new instance $\mathcal{I}_\alpha$ from $\mathcal{I}$ by multiplying all penalties by α. Let T^{GW} be the tree returned by* GW *on $\mathcal{I}_\alpha$, and let $\gamma = \frac{\pi(\overline{T^{GW}})}{OPT}$. Then $c(T^{GW}) + \pi(\overline{T^{GW}}) \leq (2 - (2\alpha - 1)\gamma - 2(1 - \alpha)\delta)OPT$.*

Proof: Let OPT_α be the cost of the optimal solution for instance $\mathcal{I}_\alpha$. Since O is one possible solution, $OPT_\alpha \leq c(O) + \alpha\pi(\overline{O})$. Thus, Theorem 1 yields

$$\begin{aligned} &c(T^{GW}) + 2\alpha\pi(\overline{T^{GW}}) \leq 2OPT_\alpha \leq 2(c(O) + \alpha\pi(\overline{O})) \\ &c(T^{GW}) + \pi(\overline{T^{GW}}) \leq 2(c(O) + \pi(\overline{O})) \\ &\qquad\qquad - (2\alpha - 1)\pi(\overline{T^{GW}}) - 2(1 - \alpha)\pi(\overline{O}) \\ &\qquad\qquad = (2 - (2\alpha - 1)\gamma - 2(1 - \alpha)\delta)OPT. \end{aligned}$$

The second line follows by subtracting the same term $(2\alpha-1)\pi(\overline{T^{GW}})$ from both sides and rearranging. The other transformations come from applying the definitions of γ and δ and applying our bound on OPT_α. ■

Thus, if γ *or* δ is at least a constant, then T^{GW} is a $(2-\epsilon)$-approximation, for some constant ϵ. This accords with the intuition that the GW algorithm deals especially well with the penalty term. The difficulties come from the tree cost, so it is instructive to see an example of what goes wrong.

Suppose we run GW on $\mathcal{I}_\alpha$ and the resulting tree T^{GW} is not already a $(2-\epsilon)$-approximation. Because this implies that both γ and δ are very small, we might hope that T^{GW} and O span approximately the same set of nodes. Thus, if we ran the RZ 1.55-approximation algorithm on the ordinary Steiner tree instance with $\overline{V(T^{GW})}$ as Steiner nodes and $V(T^{GW})$ as the terminal set (which may include some nodes not in $V(O)$), we might hope that the tree it returned would be close to a 1.55-approximation for the original PCST instance. However, the following example shows that this cannot work.

Given $k \geq 2$ and z small but positive, we construct an instance of the ordinary Steiner tree problem, cast as a special case of PCST, where all penalties are either 0 or ∞. Thus, $\mathcal{I}_\alpha = \mathcal{I}$, and necessarily $\gamma = \delta = 0$. There is a $2k$-cycle, alternating between terminals and Steiner nodes, connected by edges of cost 1. There is one "hub" Steiner node in the middle, with "spoke" edges of cost $1+z$ to each of the terminals in the cycle. In this case, T^{GW} is a path containing all cycle edges except for the pair on either side of one of the Steiner nodes, costing $2k-2$ (including all but two Steiner nodes: this single one on the cycle, and the hub node). Meanwhile, OPT buys only the k spoke edges (and, by contrast, includes only one Steiner node, the hub), at total cost $k(1 + z)$. This ratio goes to 2 as $k \to \infty$ and $z \to 0$. But in this case, T^{GW} already spans the set $V(T^{GW})$ of new terminals *optimally*, so there is nothing to be gained by calling RZ on terminal set $V(T^{GW})$ (i.e., the two nodes in $\overline{V(T^{GW})}$ do not help). A better idea is required.

3. Our PCST algorithm

Our algorithm PCST-Alg(α, β) (parametrized by constants $\alpha \in [\frac{1}{2}, 1]$, $\beta \geq 1$) is relatively simple to describe, but its analysis is quite involved. We produce two solutions to instance $\mathcal{I}$, and output the better one. As hinted in Section 2.1, the first solution is T^{GW}, the output of GW on instance $\mathcal{I}_\alpha$. For the second solution, we use GW-Growth to identify a set of terminals defining an instance of the ordinary Steiner tree problem, and run RZ to generate a second solution T^{RZ}.[4]

The bad example of Section 3 shows that we cannot identify this terminal set in the obvious way; we need to be more clever. In particular, we define another instance $\mathcal{I}_{\alpha,\beta}$ as follows. For notational simplicity, let $S = V(T^{GW})$. We derive $\mathcal{I}_{\alpha,\beta}$ from $\mathcal{I}$ by scaling *up* all penalties by a factor of β, then changing the penalty of each node in $\overline{S}$ to zero. We run GW-Growth on $\mathcal{I}_{\alpha,\beta}$, and let D be the set of all nodes in S that were *ever* part of a dead cluster during GW-Growth. (Note that all nodes in $\overline{S}$ get deactivated at time 0, but they are *not* in D.) We do not run GW-Pruning at all. The set of terminals that we send to the RZ algorithm is $S - D$. Note that choosing $S - D$ instead of S defeats the bad example from Section 2.1, because in that case D consists of the $k - 1$ Steiner nodes on the cycle that had been part of S and were causing trouble.

Recall that O denotes both the optimal tree and the set of nodes it spans, and γ and δ are defined such that $\pi(\overline{S}) = \gamma OPT$ and $\pi(\overline{O}) = \delta OPT$. Since we have control over α and β but not over γ or δ, our analysis can select the best possible α and β but must assume the worst γ and δ. Lemma 2 shows that if either γ *or* δ is at least a constant, T^{GW} already provides a $(2 - \epsilon)$-approximation. The big work is in showing that when γ and δ are *both* tiny, T^{RZ} provides a $(2 - \epsilon)$-approximation.

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. Lemma 3 allows us to bound the penalty cost $\pi(\overline{T^{RZ}})$, and Lemma 4 allows us to bound the edge cost $c(T^{RZ})$. Since T^{RZ} spans at least the terminal set $S - D$, it incurs at most $\pi(D \cup \overline{S})$ in penalty cost. Since $\pi(\overline{S}) = \gamma OPT$, we just need to bound $\pi(D)$. We first state that $\pi(D)$ can be made negligible by choosing β large enough.

Lemma 3. *For set D in* PCST-Alg(α, β)*, $\beta\pi(D) \leq c(T^{GW}) \leq 2(1 - (1 - \alpha)\delta - \alpha\gamma)OPT$.*

To bound $c(T^{RZ})$, we prove the following key (non-algorithmic) fact: starting from the optimal tree O, the *marginal* cost of extending this tree to connect to the nodes in $(S-D)-O$ is no larger than $(2\beta\delta)OPT$. This guarantees existentially that there is a tree that spans all nodes in $S-D$ and is not much more expensive than OPT; hence RZ won't pay too much when fed terminal set $S - D$.

Lemma 4. *Let $I = O \cap S$ and $A = (S - D) - O = (S - D) - I$. Then there exists a forest F with the following properties:*

1) *each node of A is included in one of the (nontrivial) trees of F,*
2) *each tree in the forest includes exactly one node from I, and*
3) *$c(F) \leq (2\beta\delta)OPT$.*

Now it's just a matter of mopping up.

Lemma 5. *The second solution T^{RZ} in* PCST-Alg(α, β) *achieves an approximation ratio of at most*

$$\rho(1 + (2\beta - 1)\delta) + \gamma + \frac{2}{\beta}(1 - (1 - \alpha)\delta - \alpha\gamma),$$

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

Proof: The edge cost of the optimal tree is $(1-\delta)OPT$, so we can augment it with the forest F from Lemma 4 to obtain a tree spanning at least $S - D$ and costing at most $(1 + (2\beta - 1)\delta)OPT$. The edge cost of T^{RZ} is at most ρ times as much. In addition, T^{RZ} pays at most $\pi(D \cup \overline{S})$ in penalties, since it spans at least the terminals $S - D$. But $\pi(\overline{S}) = \gamma OPT$ by definition, and $\pi(D) \leq \frac{2}{\beta}(1-(1-\alpha)\delta-\alpha\gamma)OPT$ by Lemma 3. ■

It is now a simple matter to prove an approximation ratio of $2 - \epsilon$.

Theorem 6. *For some appropriate α, β and some constant $\epsilon > 0$,* PCST-Alg(α, β) *is a $(2 - \epsilon)$-approximation algorithm for* PCST.

Proof: Let $B^{GW} = 2 - (2\alpha - 1)\gamma - 2(1 - \alpha)\delta$ and $B^{RZ} = \rho(1 + (2\beta - 1)\delta) + \gamma + \frac{2}{\beta}(1 - (1 - \alpha)\delta - \alpha\gamma)$ be the upper bounds on the approximation ratios of solutions T^{GW} and T^{RZ} given in Lemmas 2 and 5, respectively. We get to choose α, β, but all we know about γ and δ is that they lie in $[0, 1]$.

Take $\alpha = 0.75$, making $B^{GW} = 2 - \frac{1}{2}\gamma - \frac{1}{2}\delta$. We may assume that $\gamma, \delta \leq 0.001$, for otherwise B^{GW} is at most 1.9995. But then $B^{RZ} \leq \rho(1 + 2\beta\delta) + \gamma + \frac{2}{\beta} \leq 1.55(1 + 0.002\beta) + 0.001 + \frac{2}{\beta}$. Now take $\beta = 100$, to infer that $B^{RZ} \leq 1.55(1.2) + 0.001 + 0.02 = 1.881 < 1.9995$. ■

With a more careful analysis, we can improve the approximation bound to that given below in Theorem 7, whose proof we provide at the end of Section 3.1.

Theorem 7. *For some appropriate α, β,* PCST-Alg(α, β) *is a 1.991902-approximation algorithm for* PCST.

3.1. Proving the key lemmas

We have now reduced our main result, Theorem 7, to proving two key pieces, Lemmas 3 and 4. We start by

[4]Although T^{GW} depends on α, and T^{RZ} on both α and β, we suppress this in the notation since we view α and β as fixed.

relating $\pi(D)$ to $c(T^{GW})$ via the dual variables generated when we run GW-GROWTH on $\mathcal{I}_{\alpha,\beta}$.

Lemma 8. *Let y denote the dual solution generated by running* GW-GROWTH *on $\mathcal{I}_{\alpha,\beta}$. Then $\sum_{R \subseteq D \cup \overline{S}} y_R = \beta\pi(D)$.*

Proof: In GW-GROWTH, the initial GW-clusters are singletons, and new GW-clusters are formed only by merging old ones. Thus, the collection of GW-clusters forms a laminar family, meaning that every pair of GW-clusters is either disjoint, or one contains the other. Thus, $D \cup \overline{S}$, the set of all nodes that ever died, can be expressed as the disjoint union of the maximal dead GW-clusters, $C_1, \ldots, C_k$. But by construction, a GW-cluster dies when its penalty packing constraint becomes tight. Because C_i is dead, $\sum_{R \subseteq C_i} y_R = \pi_{\alpha,\beta}(C_i)$, where $\pi_{\alpha,\beta}$ is the penalty vector for instance $\mathcal{I}_{\alpha,\beta}$. Now $\sum_{R \subseteq D \cup \overline{S}} y_R = \sum_{\text{GW-clusters} R : R \subseteq D \cup \overline{S}} y_R$, since only a GW-cluster R can have nonzero y_R. Furthermore, each GW-cluster $R \subseteq D \cup \overline{S}$ satisfies $R \subseteq C_i$ for some i, since the GW-clusters form a laminar family and the C_i are the maximal clusters contained in $D \cup \overline{S}$. Hence $\sum_{R \subseteq D \cup \overline{S}} y_R = \sum_{i=1}^{k} \sum_{R \subseteq C_i} y_R$. Now the fact that $D \cup \overline{S}$ is the disjoint union of $C_1, C_2, \ldots, C_k$ implies that $\sum_{i=1}^{k} \pi_{\alpha,\beta}(C_i) = \pi_{\alpha,\beta}(D \cup \overline{S}) = \beta\pi(D)$. Altogether we have

$$\sum_{R \subseteq D \cup \overline{S}} y_R = \sum_{i=1}^{k} \sum_{R \subseteq C_i} y_R = \sum_{i=1}^{k} \pi_{\alpha,\beta}(C_i) = \beta\pi(D).$$

■

Let us now consider what LP (3) looks like for an instance of the ordinary Steiner tree problem. If node set $R \subseteq V - \{r\}$ contains no terminals, then the penalty packing constraints imply that $y_U = 0$ for all $U \subseteq R$, since $\pi(R) = 0$. But if R contains at least one terminal (whose penalty is infinite), then the penalty packing constraint becomes vacuous because $\pi(R) = \infty$. This leads to the following lemma.

Lemma 9. *Let $X \subseteq V$ be a set of terminals, $r \in X$, and T be any solution to the ordinary Steiner tree problem with terminal set X (i.e., any tree T spanning X). Suppose a vector y of dual variables obeys all of the edge packing constraints of LP* (3)*, but not necessarily the penalty packing constraints. Then*

$$\sum_{R \subseteq V - \{r\} : R \cap X \neq \emptyset} y_R \leq c(T). \tag{6}$$

Proof: Consider the dual LP (3) for this Steiner tree instance. Define a new vector y' by $y'_U = 0$ if $U \cap X = \emptyset$, and $y'_U = y_U$ otherwise. For any set $R \subseteq V - \{r\}$ such that $R \cap X = \emptyset$, y' satisfies the penalty packing constraint trivially because $\sum_{U \subseteq R} y'_U = 0 = \pi(R)$; when $R \cap X = \emptyset$, the constraint is trivial because $\pi(R) = \infty$.

The value of this dual solution y' is precisely $\sum_{R \subseteq V - \{r\}} y'_R = \sum_{R \subseteq V - \{r\} : R \cap X \neq \emptyset} y_R$, which lower bounds the cost of every primal solution, T in particular. ■

Lemma 3 now follows as a corollary.

Proof of Lemma 3: Consider the ordinary Steiner tree instance with terminal set $D \cup \{r\}$. Since D is defined to be a subset of S, T^{GW} spans $D \cup \{r\}$, and is hence a solution to this Steiner tree instance. By design, the dual variables y constructed by GW-GROWTH on instance $\mathcal{I}_{\alpha,\beta}$ satisfy the edge packing constraints of LP (3). Every $R \subseteq D \cup \overline{S}$ such that $y_R > 0$ excludes r and contains some node from D, since $\overline{S}$ has zero penalty in $\mathcal{I}_{\alpha,\beta}$. Thus, applying Lemma 8 then Lemma 9 with $X = D \cup \{r\}$ gives

$$\beta\pi(D) = \sum_{R \subseteq D \cup \overline{S}} y_R \leq \sum_{\substack{R \subseteq V - \{r\}: \\ R \cap (D \cup \{r\}) \neq \emptyset}} y_R \leq c(T^{GW}).$$

Lemma 2 implies $c(T^{GW}) \leq 2(1 - (1-\alpha)\delta - \alpha\gamma)OPT$ after subtracting $\pi(\overline{T^{GW}}) = \gamma OPT$ on both sides. ■

We now work on proving Lemma 4. Let $I = O \cap S$ be the intersection of the set of nodes spanned by OPT and those spanned by T^{GW}. Let $A = (S - D) - I$. Lemma 4 says that there exists an inexpensive forest F of edges connecting I to A. That is, we can span $O \cup A$ cheaply, paying OPT to span O, plus $c(F)$ to augment the optimal tree to span A as well. Therefore, running a good approximation algorithm for Steiner tree with terminal set $O \cup A$ (or any subset thereof, such as $S - D$, the subset of $O \cup A$ on which we actually run), will yield a cheap tree. For this argument, it is critical that we are solving the *rooted* PCST problem, because both S and O must contain r, guaranteeing that $I \neq \emptyset$.

We now define a set of GW-clusters that plays an important role in this lemma.

Definition 10. Let $\mathcal{G} = \{\text{GW-clusters } C : C \subseteq \overline{I}\}$.

Our strategy is to upper bound $c(F)$ in terms of $\sum_{C \in \mathcal{G}} y_C$, where the y_C are the dual variables generated by running GW-GROWTH on $\mathcal{I}_{\alpha,\beta}$. This sum must be small because of the penalty packing constraint.

Proof of Lemma 4: If $A = \emptyset$, we can take $F = \emptyset$ and the result is trivial. Otherwise, let T and y be the tree and dual solution that GW-GROWTH generates on instance $\mathcal{I}_{\alpha,\beta}$. We will derive F from T by deleting some edges in two phases. Recall that T spans the entire node set V, since we never ran GW-PRUNING. 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

$$c(F) \leq 2 \sum_{C \in \mathcal{G}} y_C. \tag{7}$$

By the penalty packing constraint, the sum is at most the total penalty of $\overline{I}$ in $\mathcal{I}_{\alpha,\beta}$, which is $\beta\pi(S - I) \leq \beta\pi(\overline{O}) =$

$\beta\delta OPT$, since $\overline{S}$ gets no penalty in $\mathcal{I}_{\alpha,\beta}$ and $S - I \subseteq \overline{O}$. Property 3 then follows.

Our description of F is constructive in the mathematical sense but not the computational sense, because it depends on the set $I = O \cap S$, and our algorithm does not know O. We now show how to construct F by deleting edges from T. To help us prove (7), we will also ensure that F satisfies the following additional property.

If u, v are distinct nodes in I, 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-GROWTH. For each such pair u, v, $e_{uv} \notin F$. (8)

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-GROWTH. When we arrive at an edge e, if there exist two nodes in I that are still in the same tree 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 I. Since I and A are disjoint, for each node in A, its tree contains a distinct node from I, and is hence non-trivial. Thus, property 1 holds throughout this pruning phase. We will satisfy (8) by the end, because at the time we consider edge e_{uv}, nodes u and v are still connected by the path P_{uv}, and so we delete e_{uv}. Thus, F will end up with exactly one node of I per tree, satisfying property 2. We must do further pruning in order to achieve (7).

Next, we prune each tree in the forest F in precisely the same way that GW-PRUNING would do, except for the definition of which GW-clusters are colored black. For our pruning rule, we color a GW-cluster C black if C died at any point during GW-GROWTH *and* $C \in \mathcal{G}$ (i.e., C contains no vertices from I). Since only the nodes in $D \cup \overline{S}$ ever died, the black GW-clusters contain only nodes from $(D \cup \overline{S}) - I$, so this pruning does not prune away any node from A or I. Moreover, since each pruning step throws away an entire subtree, the remaining part of the pruned tree is still a single component. Thus, properties 1 and 2 are maintained. The remaining forest is our final F.

We now prove that this F satisfies (7). Our proof is very similar to the famous proof of Theorem 1. We first break apart the dual variables by epoch. Let t_j be the time at which the j^{th} event point occurs in GW-GROWTH ($0 = t_0 \leq t_1 \leq t_2 \leq \cdots$), so that the j^{th} epoch is the interval of time from t_{j-1} to t_j. For each GW-cluster C, let $y_C^{(j)}$ be the amount by which y_C grew during epoch j, which is $t_j - t_{j-1}$ if it was active during this epoch, and 0 otherwise. Thus, $y_C = \sum_j y_C^{(j)}$. Because each edge e of F was added at some point by GW-GROWTH when its edge packing constraint in LP (3) became tight, we can exactly apportion the cost c_e amongst the collection of GW-clusters $\{C : e \in \delta(C)\}$ whose dual variables "pay for" the edge, and can divide this up further by epoch. In other words, $c_e = \sum_j \sum_{C: e \in \delta(C)} y_C^{(j)}$. Hence it suffices to prove that the total edge cost from F that is apportioned to epoch j is at most $2 \sum_{C \in \mathcal{G}} y_C^{(j)}$, i.e.,

$$\sum_{e \in F} \sum_{C: e \in \delta(C)} y_C^{(j)} \leq 2 \sum_{C \in \mathcal{G}} y_C^{(j)}. \tag{9}$$

In other words, during each epoch, the total rate at which edges of F are paid for by all active GW-clusters is at most twice the number of active GW-clusters just in $\mathcal{G}$. Summing over the epochs yields (9). More formally,

$$\begin{aligned} c(F) = \sum_{e \in F} c(e) &= \sum_{e \in F} \sum_j \sum_{C: e \in \delta(C)} y_C^{(j)} \\ &= \sum_j \sum_{e \in F} \sum_{C: e \in \delta(C)} y_C^{(j)} \leq \sum_j [2 \sum_{C \in \mathcal{G}} y_C^{(j)}] \\ &= 2 \sum_{C \in \mathcal{G}} \sum_j y_C^{(j)} = 2 \sum_{C \in \mathcal{G}} y_C. \end{aligned}$$

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 existed during epoch j partitions V, but we are concerned only with the partition of $V(F)$ induced by intersecting each GW-cluster with $V(F)$, and hence we care about only the GW-clusters in $\mathcal{C}_j$. Consider the graph $(V(F), F)$, and then contract 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 does. Although the GW-clusters in $\mathcal{C}_j$ are identified with nodes of H, we will continue to refer to them as GW-clusters, in order to to avoid confusion with the nodes of the original graph. During the jth epoch, some of the GW-clusters are active and some may be dead. Let us denote the sets of active and dead GW-clusters in $\mathcal{C}_j$ by $\mathcal{C}_{act}$ and $\mathcal{C}_{dead}$, respectively. The edges of F that are being partially paid for during 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 $(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_{act}} \deg_H(C)$. Since, for every active GW-cluster C in $\mathcal{G}$, $y_C^{(j)}$ grows by exactly $t_j - t_{j-1}$ in epoch j, we have $\sum_{C \in \mathcal{G}} y_C^{(j)} \geq \sum_{C \in \mathcal{G} \cap \mathcal{C}_j} y_C^{(j)} = (t_j - t_{j-1}) |\mathcal{G} \cap \mathcal{C}_{act}|$.[5] Thus, it suffices to show that $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2 |\mathcal{G} \cap \mathcal{C}_{act}|$.

Let us call the GW-clusters in $\mathcal{G} \cap \mathcal{C}_{act}$ *hungry* and the active or dead GW-clusters that intersect I *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 make some simple observations about H. Since T is a tree, F is a subset of the edges in T, and each GW-cluster represents a disjoint induced subtree of T, the contraction to H introduces no cycles. Thus, H

[5] The inequality may be strict if some $C \in \mathcal{G}$ is growing in epoch j, and hence $y_C^{(j)} > 0$, but $C \cap V(F) = \emptyset$, so $C \notin \mathcal{C}_j$.

is a forest. Every dead leaf of H must be sated, because otherwise the corresponding GW-cluster C would be black and hence would have been pruned away during GW-PRUNING. 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 I and each sated GW-cluster intersects I, but it can be the case that some $C \in \mathcal{C}_j$ contains nodes from multiple trees of F, so that when we contract C, these multiple trees become a single tree in H, and that single tree might have multiple nodes of I. We will use property (8) to show that if this happens, then the nodes in I from the distinct trees of F that were merged are actually in the same sated GW-cluster during this epoch. Then we will be almost done.

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 node from I—call these nodes u, v—and consider the path P_{uv} from u to v in T. Since u, v are distinct nodes of I, they are in different trees of F (by property 2), but they could both end up getting collapsed into the same tree of H. This can happen only if some GW-cluster C exists in epoch j that contains a node a from u's tree in F and a node b from v's tree in F. In the current epoch, in C there is a (unique) $a-b$ path in the current forest. Because there is an $a-b$ path, there is a $u-v$ path. Therefore C contains both u and v, contradicting the assumption that the cluster containing u is disjoint from the one containing v.

With this information about H, it is easy to bound $\sum_{C\in\mathcal{C}_{act}} \deg_H(C)$. Let k be the number of trees in H, and l be the number of dead sated leaves in H. Because each tree of H has exactly one node of I, there exist exactly k sated GW-clusters in $\mathcal{C}_j$ (one per tree of H). Since at least l of the sated clusters are dead, at most $k-l$ sated clusters are active. Since every active cluster is either sated or hungry,

$$(\#\text{hungry}) = (\#\text{active}) - (\#\text{active and sated})$$

and therefore

$$|\mathcal{G} \cap \mathcal{C}_{act}| \geq |\mathcal{C}_{act}| - (k - l). \tag{10}$$

The number of GW-clusters in H is $|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|$, so the number of edges is $(|\mathcal{C}_{act}|+|\mathcal{C}_{dead}|)-k$ (there being k trees in H), and hence $\sum_{C\in\mathcal{C}_{act}\cup\mathcal{C}_{dead}} \deg_H(C) = 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|-k)$. Since the only dead leaves are the l sated ones, the sum of the degrees of the dead GW-clusters is at least $2|\mathcal{C}_{dead}|-l$, since every nonleaf has degree at least 2. Thus,

$$\begin{aligned}\sum_{C\in\mathcal{C}_{act}} \deg_H(C) &\leq 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}| - k) - (2|\mathcal{C}_{dead}| - l)\\ &= 2|\mathcal{C}_{act}| - 2k + l \leq 2|\mathcal{G}\cap\mathcal{C}_{act}| - l\\ &\leq 2|\mathcal{G}\cap\mathcal{C}_{act}|,\end{aligned}$$

by (10), as we wished to show. ■

Finally, we prove the approximation bound of Theorem 7.

Proof of Theorem 7: We define B^{GW} and B^{RZ} as in the proof of Theorem 6. Choosing the best α and β and assuming the worst γ and δ gives an approximation ratio of

$$\min_{\alpha,\beta} \max_{\gamma,\delta} \min\{B^{GW}, B^{RZ}\}, \tag{11}$$

where the outer min is over $\alpha \in [\frac{1}{2}, 1], \beta \geq 1$ and the max is over $\gamma, \delta \in [0, 1]$. For any fixed α, β, the inner maximization can be written as the following LP:

$$\begin{aligned}&\text{maximize } z\\ &\text{subject to:}\\ &\quad z \leq 2 - (2\alpha - 1)\gamma - 2(1-\alpha)\delta\\ &\quad z \leq (\rho + \frac{2}{\beta}) + (1 - \frac{2\alpha}{\beta})\gamma + (\rho(2\beta - 1) - \frac{2}{\beta}(1-\alpha))\delta\\ &\quad z \geq 0, \gamma, \delta \in [0, 1]\end{aligned}$$

Using strong LP duality, we can replace this LP with its dual:

$$\begin{aligned}&\text{minimize } 2\lambda^{GW} + (\rho + \frac{2}{\beta})\lambda^{RZ}\\ &\text{subject to:}\\ &\lambda^{GW} + \lambda^{RZ} \geq 1\\ &(2\alpha - 1)\lambda^{GW} - (1 - \frac{2\alpha}{\beta})\lambda^{RZ} \geq 0\\ &2(1-\alpha)\lambda^{GW} - \left(\rho(2\beta - 1) - \frac{2}{\beta}(1-\alpha)\right)\lambda^{RZ} \geq 0\\ &\lambda^{GW}, \lambda^{RZ} \geq 0\end{aligned} \tag{12}$$

Treating α and β as variables in (12) yields a non-linear program (NLP) equivalent to (11). Every feasible solution of this NLP yields a valid upper bound on (11). Using a numeric solver, we find a local minimum at $\alpha = 0.516513$, $\beta = 8.88105$, $\lambda^{GW} = 0.963974$, $\lambda^{RZ} = 0.036026$, giving an approximation ratio < 1.991902. ■

4. PRIZE-COLLECTING TSP AND STROLL

Our algorithm PCTSP-ALG(α, β) for PCTSP is nearly identical to PCST-ALG(α, β), with some slight differences that we highlight in a moment, but first we discuss the algorithm GW-PCTSP given by Goemans and Williamson [18]. The LP relaxation that undergirds their algorithm for PCTSP is the same as (2), except that there is a coefficient of 2 on the z variables and right side of the primal covering constraint. Hence in the dual LP (3), there is a coefficient of 2 in front of the objective function and the y variables in the penalty packing constraint. GW-PCTSP multiplies all penalties by $\frac{1}{2}$, runs the GW algorithm for PCST, doubles all of the edges of the resulting tree T to make all node degrees even, finds an Eulerian tour, and shortcuts it to obtain a cycle. Recall that we assume that the costs satisfy the triangle inequality, so this last step does not increase the cost. This algorithm achieves the same guarantee as that stated for GW in Theorem 1, i.e., it is a Lagrangian-preserving 2-approximation. The analysis, in short, is that even though the cycle cost may be as much as twice the tree cost, we can afford to pay for it because of the factor of 2 in the dual objective function. The penalties were multiplied by $\frac{1}{2}$ to prevent GW from violating the amended penalty packing constraint, so the pruned components are exactly paid for by twice their dual variables, which is okay again because of the factor of 2 in the dual objective.

We now describe and analyze PCTSP-ALG(α, β). To generate the first solution, we simply run GW-PCTSP on $\mathcal{I}_\alpha$ (i.e., in GW-PCTSP, GW-GROWTH is run on $\mathcal{I}_{\frac{1}{2}\alpha}$). Since Lemma 1 still holds, so does Lemma 2. To generate the second solution, we run GW-GROWTH on $\mathcal{I}_{\frac{1}{2}\alpha,\frac{1}{2}\beta}$, define the sets S and D as before, and then run the Christofides $\frac{3}{2}$-approximation algorithm for TSP [9] on the set $S - D$. By similar reasoning as above, we can still show that $\pi(D) \leq 2(1-(1-\alpha)\delta - \alpha\gamma)OPT$ as in Lemma 3, and Lemma 4 remains the same, except that we get the stronger bound $c(F) \leq \beta\delta OPT$, because our upper bound on the penalty of $\overline{I}$ is now $\frac{1}{2}\beta\delta OPT$, half what it was in PCST-ALG(α, β). This is fortunate, because this shows that if we double each edge of F, add it to the optimal tour O, find an Eulerian tour and shortcut it, we will have produced a tour visiting $S-D$ of edge cost at most $(1+(2\beta-1)\delta)OPT$, as before. Thus, the bound on the cost of our second solution is the same as in Lemma 5, except with $\rho = \frac{3}{2}$ (from using Christofides instead of RZ).

Our algorithm for PCS is similar, except that we use the Lagrangian-preserving 2-approximation of Chaudhuri et al. for PCS [8] in place of GW-PCTSP, and the $\frac{3}{2}$-approximation of Hoogeveen for the path version of TSP [20] in place of Christofides. Some care must be taken to deal with the fact that the Chaudhuri et al. algorithm guesses the tail end of the optimal stroll, and the dual variables y_C for sets C containing this tail have coefficient 1 (not 2) in the dual objective function, but in the end we achieve the same approximation ratio as in Lemma 5.

This gives the following theorem.

Theorem 11. *For some appropriate α, β, algorithms* PCTSP-ALG(α, β) *and* PCS-ALG(α, β) *yield an approximation ratio of 1.989691 for* PCTSP *and* PCS*, respectively.*

Proof: The proof is the same as for Theorem 7, except that $\rho = \frac{3}{2}$, so we choose $\alpha = 0.518719$, $\beta = 7.99878$, $\lambda^{GW} = 0.958757$ and $\lambda^{RZ} = 0.041243$. ■

5. EVOLVING VIEWS OF THE GOEMANS-WILLIAMSON ALGORITHM

The history of Theorem 1 is highly tangled, so we tease apart the strands here.

The paper by Goemans and Williamson [18] proposes a slightly different version of algorithm GW, which we will refer to as GW-ORIG, consisting of two phases, GW-GROWTH-ORIG and GW-PRUNING-ORIG. GW-GROWTH-ORIG is the same as GW-GROWTH, except that they make the root cluster always inactive, because the fake dual variables that we introduced for these clusters do not actually appear in the LP from Section 2, so they cannot contribute to the dual objective function. The growth phase terminates once there are no more active components; thus, the forest it returns may contain more than one tree, in the event that some cluster died and never merged with the root cluster. In GW-PRUNING-ORIG, all trees are immediately discarded except for the one containing r. When pruning this one tree, the root clusters are not colored black (despite being always inactive), and hence are never pruned. In Theorem 4.1 of [18], they prove that the tree T returned by their algorithm satisfies the following guarantee:

$$c(T) + \pi(\overline{T}) \leq \left(2 - \frac{1}{n-1}\right) OPT, \tag{13}$$

where $n = |V|$. Later, Chudak, Roughgarden and Williamson [10] observed that the same algorithm actually provides the stronger Lagrangian-preserving guarantee

$$c(T) + (2 - \frac{1}{n-1})\pi(\overline{T}) \leq \left(2 - \frac{1}{n-1}\right) OPT. \tag{14}$$

This appears as Theorem 2.1 in [10] without proof, because the result is already implicit in the proof of Theorem 4.1 from [18]; the original paper simply failed to claim the stronger result.

Johnson, Minkoff and Phillips [23] first suggested the variant of GW that we present in Section 2, and claim falsely in their Theorem 3.2 that it achieves the same guarantee (13) as the original version GW-ORIG in [18]. However, Feofiloff et al. [12] give a counterexample to show that this variant achieves an approximation ratio no better than 2. In an unpublished manuscript [11], the same authors show that the approximation ratio for this variant is indeed 2. In other words, they prove in their Theorem 4.1 that the tree T resulting from GW-GROWTH followed by GW-PRUNING satisfies

$$c(T) + \pi(\overline{T}) \leq 2OPT, \tag{15}$$

by slightly tweaking the original proof from [18]. In fact, they actually prove the stronger, Lagrangian-preserving guarantee that we state in Theorem 1, but fail to claim the stronger result. In [12], these authors propose another variant of GW, which is based on a slightly different LP relaxation (directly addressing the unrooted version of PCST), and prove that it achieves a slightly better approximation ratio of $2 - \frac{2}{n}$. Again, the solution generated by their improved algorithm actually satisfies the stronger Lagrangian-preserving guarantee, but they fail to claim the stronger result.

In our main algorithm of Section 3, we run GW-GROWTH twice. First we run it on instance $\mathcal{I}_\alpha$, followed by GW-PRUNING, to generate a solution T^{GW}. Second, we run

GW-GROWTH on instance $\mathcal{I}_{\alpha,\beta}$ to figure out which set of terminals to pass to the RZ algorithm, which then generates solution T^{RZ}. In the second case, it is critical that we run GW-GROWTH and *not* GW-GROWTH-ORIG, because we use the fact that the tree output by GW-GROWTH spans all of V. However, in the first case, we could just as well have used GW-ORIG instead of GW.

We chose to use GW-GROWTH in both cases only because we thought it made for a cleaner presentation. For any readers who are uncomfortable that our Theorem 1 relies on piecing together both published ([18], [10]) and unpublished ([11]) work, we offer two observations. First, Theorem 1 really requires only a tiny modification to the proof of Theorem 4.1 in [18]. Second, if we amend PCST-ALG to produce T^{GW} by using GW-ORIG instead of GW, then our main result, Theorem 7, can be made to be independent of the unpublished manuscript [11], relying instead on property (14), which was published in [10].

6. FUTURE WORK

Our techniques might be applicable to the PCST problem with submodular penalties, introduced in [19].

REFERENCES

[1] A. Agrawal, P. N. Klein, and R. Ravi, "When trees collide: An approximation algorithm for the generalized Steiner problem on networks," *SIAM J. on Computing*, 24:440–456, 1995.

[2] A. Archer, A. Levin, and D. P. Williamson, "A faster, better approximation algorithm for the minimum latency problem," *SIAM J. on Computing*, 37: 1472–1498, 2008.

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

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

[5] E. Balas, "The prize collecting traveling salesman problem," *Networks*, 19: 621–636, 1989.

[6] P. Berman and V. Ramaiyer, "Improved approximations for the Steiner tree problem," in *Proceedings of SODA 1992*, 325–334.

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

[8] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, "Paths, trees, and minimum latency tours," in *Proceedings of FOCS 2003*, 36–45.

[9] N. Christofides, "Worst-case analysis of a new heuristic for the travelling-salesman problem," Graduate School of Industrial Administration, Carnegie-Mellon University, Tech. Rep., 1976.

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

[11] P. Feofiloff, C. G. Fernandes, C. E. Ferreira, and J. C. de Pina, "A note on Johnson, Minkoff and Phillips' algorithm for the prize-collecting Steiner tree problem," 2006, manuscript available from http://www.ime.usp.br/ cris/publ/jmp-analysis.ps.gz.

[12] ——, "Primal-dual approximation algorithms for the prize-collecting Steiner tree problem," *Information Processing Letters*, 103: 195–202, 2007.

[13] N. Garg, "A 3-approximation for the minimum tree spanning k vertices," in *Proceedings of FOCS 1996*, 302–309.

[14] ——, "Saving an epsilon: a 2-approximation for the k-MST problem in graphs," in *Proceedings of STOC 2005*, 396–402.

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

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

[17] M. X. Goemans and D. P. Williamson, "A general approximation technique for constrained forest problems," in *Proceedings of SODA 1992*, 307–316.

[18] ——, "A general approximation technique for constrained forest problems," *SIAM J. on Computing*, 24: 296–317, 1995.

[19] A. Hayrapetyan, C.Swamy and É. Tardos, "Network design for information networks," *Proceedings of SODA 2005*, 933-942.

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

[21] S. Hougardy and H. J. Prömel, "A 1.598 approximation algorithm for the Steiner problem in graphs," in *Proceedings of SODA 1999*, 448–453.

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

[23] D. S. Johnson, M. Minkoff, and S. Phillips, "The prize collecting Steiner tree problem: theory and practice." in *Proceedings of SODA 2000*, 760–769.

[24] M. Jünger and W. R. Pulleyblank, "New primal and dual matching heuristics," *Algorithmica*, 13: 357–386, 1995.

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

[26] H. J. Prömel and A. Steger, "RNC-approximation algorithms for the Steiner problem," in *Proceedings of STACS 1997*, 559–570.

[27] G. Robins and A. Zelikovsky, "Tighter bounds for graph Steiner tree approximation." *SIAM J. Discrete Mathematics*, 19: 122–134, 2005.

[28] A. Zelikovsky, "An 11/6-approximation for the Steiner problem on graphs," in *Proceedings of the Czechoslovakian Symposium on Combinatorics, Graphs, and Complexity (1990)* published in *Annals of Discrete Mathematics*, 51: 351–354, 1992.

[29] ——, "An 11/6-approximation algorithm for the network Steiner problem," *Algorithmica*, 9: 463–470, 1993.

[30] ——, "Better approximation bounds for the network and Euclidean Steiner tree problems," University of Virginia, Charlottesville, VA, USA, Tech. Rep., 1996.