Primal-Dual Approximation Algorithms for the Prize-Collecting Steiner Tree Problem[*]

Paulo Feofiloff[†] Cristina G. Fernandes[†] Carlos E. Ferreira[†] José Coelho de Pina[†]

January 29, 2013

Abstract

The primal-dual scheme has been used to provide approximation algorithms for many problems. Goemans and Williamson gave a $(2-\frac{1}{n-1})$-approximation for the Prize-Collecting Steiner Tree Problem that runs in $O(n^3 \log n)$ time. Their algorithm applies the primal-dual scheme once for each of the n vertices of the graph. We present a primal-dual algorithm that runs in $O(n^2 \log n)$, as it applies this scheme only once, and achieves the slightly better ratio of $2-\frac{2}{n}$. We also show a tight example for the analysis of the algorithm and discuss briefly a couple of other algorithms described in the literature.

Keywords: approximation algorithms; primal-dual method; prize-collecting Steiner tree

1 Introduction

The Prize-Collecting Steiner Tree Problem is an extension of the Steiner Tree Problem where each vertex left out of the tree pays a penalty. The goal is to find a tree that minimizes the sum of its edge costs and the penalties for the vertices left out of the tree. The problem has applications in network design and has been used to approximate other problems, such as the k-Steiner tree (finding a minimum tree spanning k vertices) and the survivable network problem [2, 8].

The best approximation algorithms known for the Prize-Collecting Steiner Tree Problem are based on the primal-dual scheme. Different linear programming formulations of the problem may lead to different algorithms. In this paper we present one such formulation and use it in the design of a new approximation algorithm.

Consider a graph $G = (V, E)$, a function c from E to $\mathbb{Q}_{\geq}$ and a function π from V to $\mathbb{Q}_{\geq}$. (Here, $\mathbb{Q}_{\geq}$ denotes the set of non-negative rationals.) The **Prize-Collecting Steiner Tree (PCST) Problem** consists of the following: given G, c, and π, find a tree T in G such that

$$\sum_{e \in E_T} c_e + \sum_{v \in V \setminus V_T} \pi_v$$

is minimum. (We denote by V_H the vertex set and by E_H the edge set of a graph H.) The **rooted** variant of the problem requires T to contain a given root vertex.

[*] This is a corrected and improved version of a paper by the same auhors [5] published in 2007.

[†] Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508-090 São Paulo/SP, Brazil. E-mail: `{pf,cris,cef,coelho}@ime.usp.br`. Research supported in part by PRONEX/CNPq 664107/1997-4 (Brazil).

Goemans and Williamson [10, 11] used the primal-dual scheme to derive a $(2 - \frac{1}{n-1})$-approximation for the rooted PCST, where $n := |V|$. Trying all possible choices for the root, they obtained a $(2 - \frac{1}{n-1})$-approximation for the (unrooted) PCST. The resulting algorithm, which we call GW, runs in time $O(n^3 \log n)$. Johnson, Minkoff and Phillips [12] proposed a variant of the algorithm that runs the primal-dual scheme only once, resulting in a $O(n^2 \log n)$ time bound. They claimed that their variant, which we refer to as JMP, achieves the same approximation ratio as algorithm GW. Unfortunately, their claim does not hold, as we show below. Cole *et al.* [3] proposed a faster implementation of the GW algorithm, which also runs the primal-dual scheme only once and produces a $(2 + 1/\text{poly}(n))$-approximation for the (unrooted) PCST.

This paper contains two results. The main one is a modification of the GW algorithm based on a somewhat different linear program. We show that this new algorithm achieves a ratio of $2 - \frac{2}{n}$. It requires only one run of the primal-dual scheme, resulting in a $O(n^2 \log n)$ time bound. We also present a family of graphs which proves that the given analysis is tight. The second result is an example where the JMP algorithm achieves an approximation ratio of only 2, which shows that the $2 - \frac{1}{n-1}$ ratio claimed by Johnson, Minkoff and Phillips does not hold.

Our new algorithm slightly improves the best known approximation ratio for the PCST problem. Though the improvement is small, the algorithm seems interesting and might be used in the design of a better algorithm. Its behavior is not very different from that of JMP, but it stops earlier, thereby achieving a better ratio.

The paper is organized as follows. The next section introduces some notation and preliminaries. Section 3 describes the new algorithm. The analysis of the algorithm is given in Sections 4 and 5. Section 6 comments on the running time of the algorithm. Section 7 discusses the JMP algorithm and its variant for the rooted PCST. The so-called pruning phase of the algorithms is briefly discussed in Section 8.

2 Notation and preliminaries

For any subset F of E, let $c(F) := \sum_{e \in F} c_e$. For any subset S of V, let $\pi(S) := \sum_{v \in S} \pi_v$. If T is a subgraph of G, we shall abuse notation and write $\pi(T)$ and $\pi(\overline{T})$ to mean $\pi(V_T)$ and $\pi(\overline{V_T})$ respectively. (Of course, $\overline{V_T}$ denotes $V \setminus V_T$.) Similarly, we shall write $c(T)$ to mean $c(E_T)$. Hence, the goal of $\text{PCST}(G, c, \pi)$ is to find a tree T in G such that $c(T) + \pi(\overline{T})$ is minimum.

For any subset L of V, let δL stand for the set of edges of G with one end in L and the other in $\overline{L}$. Given a subgraph H of G, let $\delta_H L := E_H \cap \delta L$. For any collection $\mathcal{L}$ of subsets of V and any e in E, let

$$\mathcal{L}(e) := \{L \in \mathcal{L} : e \in \delta L\}.$$

For any subset S of V and any collection $\mathcal{L}$ of subsets of V, let

$$\mathcal{L}[S] := \{L \in \mathcal{L} : L \subseteq S\} \quad \text{and} \quad \mathcal{L}_S := \{L \in \mathcal{L} : L \supseteq S\}.$$

A collection $\mathcal{L}$ of subsets of V is **laminar** if, for any two elements L_1 and L_2 of $\mathcal{L}$, either $L_1 \cap L_2 = \emptyset$ or $L_1 \subseteq L_2$ or $L_1 \supseteq L_2$. The collection of all maximal elements of $\mathcal{L}$ shall be denoted by $\mathcal{L}^*$ and the union of all sets in $\mathcal{L}$ by $\bigcup \mathcal{L}$. If $\mathcal{L}$ is laminar, the elements of $\mathcal{L}^*$ are pairwise disjoint. If, in addition, $\bigcup \mathcal{L} = V$ then $\mathcal{L}^*$ is a partition of V.

Let $\mathcal{L}$ be a laminar collection of subsets of V and y is a function from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$. For any subcollection $\mathcal{L}'$ of $\mathcal{L}$, let $y(\mathcal{L}') := \sum_{L\in\mathcal{L}'} y_L$. We say that y **respects** c if

$$y(\mathcal{L}(e)) \;\le\; c_e \tag{1}$$

for each e in E. We say that e is **tight for** y if equality holds in (1). We say that y **respects** π if

$$y(\mathcal{L}[S]) \;\le\; \pi(S) \quad \text{and} \quad y(\mathcal{L}[\overline{S}]) + y(\mathcal{L}_S) \;\le\; \pi(\overline{S}) \tag{2}$$

for each S in $\mathcal{L}$. We shall refer to the first set of inequalities as (2a) and to the second set as (2b). (The usual primal-dual scheme for the PCST problem uses only (2a).) If equality holds in (2a), we say that y **saturates** S. If equality holds in (2b), we say that y **saturates the complement** of S.

Lemma 2.1 *Given a laminar collection $\mathcal{L}$ of subsets of V and a function y from $\mathcal{L}$ into $\mathbb{Q}_\ge$, if y respects c and π then $y(\mathcal{L}) \le c(T) + \pi(\overline{T})$ for any connected subgraph T of G.*

Proof. Let S be a minimal set in $\mathcal{L}$ containing V_T. If there is no such set, let $S := V$. For $\mathcal{B} := \{L \in \mathcal{L} : \delta_T L \ne \emptyset\}$,

$$y(\mathcal{B}) \;\le\; \sum_{L\in\mathcal{B}} |\delta_T L|\, y_L \;=\; \sum_{e\in E_T} y(\mathcal{L}(e)) \;\le\; \sum_{e\in E_T} c_e \;=\; c(T)$$

by virtue of (1). For $\mathcal{C} := \mathcal{L}[\overline{S}] \cup \mathcal{L}_S$, by virtue of (2b),

$$y(\mathcal{C}) \;=\; y(\mathcal{L}[\overline{S}]) + y(\mathcal{L}_S) \;\le\; \pi(\overline{S})\,.$$

For $\mathcal{D} := \mathcal{L}[S \setminus V_T]$,

$$y(\mathcal{D}) \;=\; \sum_{X\in\mathcal{D}^*} y(\mathcal{L}[X]) \;\le\; \sum_{X\in\mathcal{D}^*} \pi(X) \;\le\; \pi(S\setminus V_T)$$

due to (2a). Since $\mathcal{L} = \mathcal{B}\cup\mathcal{C}\cup\mathcal{D}$, we have $y(\mathcal{L}) \le c(T) + \pi(\overline{S}) + \pi(S\setminus V_T) \le c(T) + \pi(\overline{T})$, as claimed. ∎

We denote by $\mathrm{opt}(G, c, \pi)$ the value of an (optimum) solution of $\mathrm{PCST}(G, c, \pi)$, i.e., the minimum value of the expression $c(T) + \pi(\overline{T})$ over all trees T of G. Lemma 2.1 has the following consequence:

Corollary 2.2 *For any laminar collection $\mathcal{L}$ of subsets of V and any function y from $\mathcal{L}$ into $\mathbb{Q}_\ge$, if y respects c and π then $y(\mathcal{L}) \le \mathrm{opt}(G, c, \pi)$.* ∎

This lower bound serves as motivation for the new algorithm.

Linear programming relaxation

The definitions of "y respects c" and "y respects π" and the lower bound on $\mathrm{opt}(G, c, \pi)$ given by Corollary 2.2 are based on the following relaxation of the PCST problem: find vectors x and z that

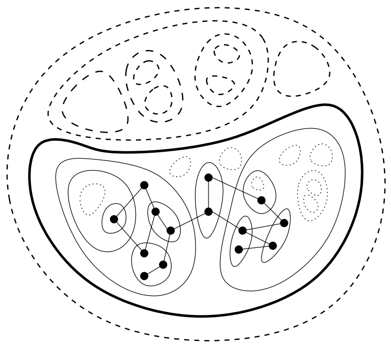

Figure 1: A connected graph T and a laminar collection of sets whose union contains V_T. The thick line encloses the set S in the proof of Lemma 2.1. The dotted lines represent the collection $\mathcal{D}$, the dashed ones together with set S represent $\mathcal{C}$ and the solid ones represent $\mathcal{B}$.

$$
\begin{array}{rrcll}
\text{minimize} & \sum_{e\in E} c_e x_e + \sum_{S\subseteq V} \pi(S) z_S & & & \\
\text{subject to} & \sum_{e\in\delta L} x_e + \sum_{S\supseteq L} z_S + \sum_{S\supseteq \overline{L}} z_S & \geq & 1 & \text{for each } L\subseteq V, \\
 & x_e & \geq & 0 & \text{for each } e\in E, \\
 & z_S & \geq & 0 & \text{for each } S\subseteq V.
\end{array} \tag{3}
$$

The idea behind this linear program is simple: in any solution given by x and z, for each set L, either there is an edge in δL, or one of L and $\overline{L}$ is part of a set that pays penalty (a set S with $z_S = 1$). Given a solution T of $\mathrm{PCST}(G, c, \pi)$, set $x_e := 1$ if $e \in E_T$ and $x_e := 0$ otherwise and set $z_S := 1$ if $S = \overline{V_T}$ and $z_S := 0$ otherwise. The pair (x, z) thus defined is feasible in (3) and its value is $c(T) + \pi(\overline{T})$. Hence, the linear program (3) is a relaxation of $\mathrm{PCST}(G, c, \pi)$. Conversely, it can be shown that any optimum solution (x, z) of (3) whose components are all in $\{0, 1\}$ describes a solution of $\mathrm{PCST}(G, c, \pi)$. The dual of the linearprogram (3) calls for a vector y that

$$
\begin{array}{rrcll}
\text{maximizes} & \sum_{L\subseteq V} y_L & & & \\
\text{subject to} & \sum_{\delta L\,\ni\, e} y_L & \leq & c_e & \text{for each } e\in E, \\
 & \sum_{L\subseteq S} y_L + \sum_{L\supseteq \overline{S}} y_L & \leq & \pi(S) & \text{for each } S\subseteq V, \\
 & y_L & \geq & 0 & \text{for each } L\subseteq V
\end{array} \tag{4}
$$

Our definition of "y respects π" follows from the second group of constraints in (4). It may seem at first that (2a) does not match that group of constraints. The apparent disagreement comes from the fact that a laminar collection does not usually contain the complements of its elements, and therefore $\mathcal{L}_{\overline{S}}$ is usually empty for S in $\mathcal{L}$. It is true, however, that, for any function y defined on a laminar collection $\mathcal{L}$, if y respects π then y also satisfies the second group of constraints in (4) once y_L is set to 0 for all L not in $\mathcal{L}$. (The proof of this observation is similar to the proof Lemma 2.1.) Corollary 2.2 is, therefore, a consequence of the weak duality relation between feasible solutions of (4) and feasible solutions of (3).

3 Unrooted growth clustering algorithm

Our algorithm, which we call GW-Unrooted-Growth, follows the GW primal-dual scheme. Just as GW, it has two phases. In phase 1, a tree is obtained; in phase 2, some edges are removed from the tree to produce the final output.

Each iteration in phase 1 begins with a spanning forest F of $G = (V, E)$, a nonnegative function y defined on the componentes of F and on some subsets of those componentes, and a collection $\mathcal{S}$ of components of F that are saturated by y. If all the components of F except one are in $\mathcal{S}$, phase 1 ends and phase 2 begins with that only component as starting point. Now suppose that two or more of the components of F are not in $\mathcal{S}$. Then the algorithm increases uniformly the values of the y variables on all those components and stops when one of the following three events happens: (a) an edge connecting two components of F becomes tight, (b) a component becomes saturated, (c) the complement of a component becomes saturated. If event (c) happens for some component T, phase 2 begins with T as starting point. If event (a) happens, the algorithm adds to F one of the edges that became tight and begins a new iteration. If event (b) happens for some component, the algorithm adds that component to $\mathcal{S}$ and begins a new iteration.

Phase 2 of our algorithm begins with the tree T produces by phase 1 and the collection $\mathcal{S}$ of saturated sets. While there exists S in $\mathcal{S}$ such that δS contains exactly one edge of T, that set S is removed from T. (This is somewhat different from the reverse delete strategy of the GW pruning algorithm.) Phase 2 returns the remaining tree.

The detailed description of our algorithm uses the following terminology. An edge is **internal to** a partition $\mathcal{P}$ of V if both of its ends are in the same element of $\mathcal{P}$. All other edges are **external to** $\mathcal{P}$. For any external edge, there are two elements of $\mathcal{P}$ containing its ends; we call these two elements the **extremes** of the edge in $\mathcal{P}$.

We can now state the GW-Unrooted-Growth algorithm. It receives (G, c, π) and returns a tree T in G such that

$$c(T) + \pi(\overline{T}) \;\le\; \left(2 - \tfrac{2}{n}\right) \mathrm{opt}(G, c, \pi)$$

as well as $c(T) + 2\,\pi(\overline{T}) \;\le\; 2\,\mathrm{opt}(G, c, \pi)$.

```
GW-Unrooted-Growth $(G, c, \pi)$
1  $F \leftarrow (V, \emptyset)$
2  $\mathcal{L} \leftarrow \{\{v\} : v \in V\}$
3  $\mathcal{S}, \mathcal{M}, y \leftarrow \emptyset, \emptyset, 0$
```

```
 4  while $|\mathcal{L}^* \setminus \mathcal{S}| > 1$ and $\mathcal{M} = \emptyset$ do
 5     let $g$ be the characteristic function of $\mathcal{L}^* \setminus \mathcal{S}$ relative to $\mathcal{L}$
 6     choose the largest $\epsilon$ in $\mathbb{Q}_{\geq}$ such that $y' := y + \epsilon g$ respects $c$ and $\pi$
 7     if some edge $e$ external to $\mathcal{L}^*$ is tight for $y'$
 8        then let $L_1$ and $L_2$ be the extremes of $e$ in $\mathcal{L}^*$
 9              $\mathcal{L} \leftarrow \mathcal{L} \cup \{L_1 \cup L_2\}$
10              $y'_{L_1 \cup L_2} \leftarrow 0$
11              $F \leftarrow F + e$
12        else if $y'$ saturates some element $L$ of $\mathcal{L}^* \setminus \mathcal{S}$
13                then $\mathcal{S} \leftarrow \mathcal{S} \cup \{L\}$
14                else $y'$ saturates the complement of an element $M$ of $\mathcal{L}$
15                     $\mathcal{M} \leftarrow \{M\}$
16     $y \leftarrow y'$
17  if $\mathcal{M} \neq \emptyset$
18     then let $M$ be the only element of $\mathcal{M}$
19     else let $M$ be the only element of $\mathcal{L}^* \setminus \mathcal{S}$
20  $T \leftarrow F[M]$
21  $\triangleright$ end of phase 1 and beginning of phase 2
21  $\mathcal{R} \leftarrow \emptyset$
22  while $|\delta_T S| = 1$ for some $S$ in $\mathcal{S}$ do
23     $\mathcal{R} \leftarrow \mathcal{R} \cup \{S\}$
24     $T \leftarrow T - S$
25  return $T$
```

(On lines 21–24, the collection $\mathcal{R}$ is, of course, redundant, but it will be useful for the analysis.) With each iteration of the block of lines 5–16, the size of $\mathcal{L}^* \setminus \mathcal{S}$ decreases by exactly 1 unit. Hence, if $|\mathcal{L}^* \setminus \mathcal{S}| \leq 1$ on line 4, then, on line 19, collection $\mathcal{L}^* \setminus \mathcal{S}$ has a single element.

Before discussing the algorithm, we must define some additional concepts and notation. For a graph H and a set S of vertices of H, we denote by $H[S]$ the subgraph of H induced by S and by $H - S$ the graph $H[\overline{S}]$.

Given a forest F in G and a subset L of V, we say that F is **L-connected** if $V_F \cap L = \emptyset$ or the induced subgraph $F[V_F \cap L]$ is connected. In other words, F is L-connected if the following property holds: for any two vertices u and v of F in L, there exists a path from u to v in F and that path never leaves L. If F spans G (as is the case during the first phase of the algorithm), the condition "$F[V_F \cap L]$ is connected" can, of course, be replaced by "$F[L]$ is connected".

For any collection $\mathcal{L}$ of subsets of V, we shall say that F is **$\mathcal{L}$-connected** if F is L-connected for each L in $\mathcal{L}$.

For any collection $\mathcal{S}$ of subsets of V, we say a tree T **has no bridge in** $\mathcal{S}$ if $|\delta_T S| \neq 1$ (therefore, either $\delta_T S = \emptyset$ or $|\delta_T S| \geq 2$) for all S in $\mathcal{S}$.

Note that each iteration of the **while** loop in phase 2 begins with an $\mathcal{L}$-connected tree T. At the end, the tree T is $\mathcal{L}$-connected and has no bridge in $\mathcal{S}$.

At the beginning of each iteration of phase 1, F is a spanning forest of G, $\mathcal{L}$ is a laminar collection of subsets of V such that $\bigcup \mathcal{L} = V$, $\mathcal{S}$ is a subcollection of $\mathcal{L}$, $\mathcal{M}$ is a subcollection of $\mathcal{L}$, and y is a function from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$. The following invariants hold at the beginning of each iteration of the block of lines 5–16:

(i1) F is $\mathcal{L}$-connected;

(i2) y respects c and π;

(i3) each edge of F is tight for y;

(i4) y saturates each element of $\mathcal{S}$;

(i5) $|\mathcal{M}| \leq 1$ and y saturates the complement of each element of $\mathcal{M}$;

(i6) for any tree T in G, if T is $\mathcal{L}$-connected and has no bridge in $\mathcal{S}$, then

$$\sum_{e \in E_T} y(\mathcal{L}(e)) + 2\,y(\mathcal{L}[\overline{T}]) + 2\,y(\mathcal{L}_T) \;\leq\; 2\,y(\mathcal{L}) \tag{5}$$

as well as $\sum_{e \in E_T} y(\mathcal{L}(e)) + y(\mathcal{L}[\overline{T}]) + y(\mathcal{L}_T) \;\leq\; \left(2 - \frac{2}{n}\right) y(\mathcal{L})$.

Here, and in what follows, $\mathcal{L}[\overline{T}]$ and $\mathcal{L}_T$ stand for $\mathcal{L}[\overline{V_T}]$ and $\mathcal{L}_{V_T}$ respectively.

4 Proof of the invariants

At the beginning of each iteration of the block of lines 5–16, invariants (i1) to (i5) clearly hold. The proof of (i6) depends on the following lemma.

Lemma 4.1 *Let T be a tree in $G = (V, E)$. Let $\mathcal{P}$ be a partition of V and $(\mathcal{A}, \mathcal{B})$ a bipartition of $\mathcal{P}$. If T is $\mathcal{P}$-connected, has no bridge in $\mathcal{B}$, and $\mathcal{P}_T = \emptyset$, then*

$$\sum_{A \in \mathcal{A}} |\delta_T A| + 2\,|\mathcal{A}[\overline{T}]| \;\leq\; 2\,|\mathcal{A}| - 2\,. \tag{6}$$

Proof. Let us say that two elements of $\mathcal{P}$ are **adjacent** if there is an edge of T with these two elements as extremes. This adjacency relation defines a graph $\mathcal{H}$ having $\mathcal{P}$ for set of vertices. Since T has no cycles and is $\mathcal{P}$-connected, the edges of $\mathcal{H}$ are in one-to-one correspondence with the edges of T that are external to $\mathcal{P}$. Hence, the degree of every vertex P of $\mathcal{H}$ is exactly $|\delta_T P|$, and therefore $\frac{1}{2}\sum_{P \in \mathcal{P}} |\delta_T P| = |E_{\mathcal{H}}|$.

Of course $\mathcal{H}$ has at least $|\mathcal{P}[\overline{T}]|$ isolated vertices and therefore at least $|\mathcal{P}[\overline{T}]| + 1$ components. Since T has no cycles and is $\mathcal{P}$-connected, $\mathcal{H}$ is a forest. Hence $|E_{\mathcal{H}}| \leq |\mathcal{P}| - |\mathcal{P}[\overline{T}]| - 1$ and therefore

$$\frac{1}{2}\sum_{P \in \mathcal{P}} |\delta_T P| \leq |\mathcal{P}| - |\mathcal{P}[\overline{T}]| - 1\,.$$

Now consider the vertices of $\mathcal{H}$ that are in $\mathcal{B}$. Since $\mathcal{B}_T = \emptyset$, we have $|\delta_T B| \geq 1$ for each B in $\mathcal{B} \setminus \mathcal{B}[\overline{T}]$. Actually, $|\delta_T B| \geq 2$ for each B in $\mathcal{B} \setminus \mathcal{B}[\overline{T}]$, since T has no bridge in $\mathcal{B}$. Hence,

$$\sum_{B \in \mathcal{B}} |\delta_T B| \;=\; \sum_{B \in \mathcal{B} \setminus \mathcal{B}[\overline{T}]} |\delta_T B| \;\geq\; 2\,|\mathcal{B}| - 2\,|\mathcal{B}[\overline{T}]|\,.$$

Finally,

$$\begin{aligned}
\sum_{A \in \mathcal{A}} |\delta_T A| \;&=\; \sum_{P \in \mathcal{P}} |\delta_T P| - \sum_{B \in \mathcal{B}} |\delta_T B| \\
&\leq\; 2\,|\mathcal{P}| - 2\,|\mathcal{B}| - 2\,|\mathcal{P}[\overline{T}]| + 2\,|\mathcal{B}[\overline{T}]| - 2 \\
&=\; 2\,|\mathcal{A}| - 2\,|\mathcal{A}[\overline{T}]| - 2\,,
\end{aligned}$$

as claimed. ∎

Corollary 4.2 *Let T be a tree in $G = (V, E)$. Let $\mathcal{P}$ be a partition of V and $(\mathcal{A}, \mathcal{B})$ a bipartition of $\mathcal{P}$. If T is $\mathcal{P}$-connected and has no bridge in $\mathcal{B}$ then*

$$\sum_{A \in \mathcal{A}} |\delta_T A| + 2|\mathcal{A}[\overline{T}]| + 2|\mathcal{A}_T| \;\le\; 2\,|\mathcal{A}|\,. \tag{7}$$

Proof. Suppose first that $\mathcal{P}_T = \emptyset$, whence also $\mathcal{A}_T = \emptyset$. Then, by Lemma 4.1, we have

$$\sum_{A \in \mathcal{A}} |\delta_T A| + 2\,|\mathcal{A}[\overline{T}]| + 2\,|\mathcal{A}_T| = \sum_{A \in \mathcal{A}} |\delta_T A| + 2\,|\mathcal{A}[\overline{T}]| \le 2\,|\mathcal{A}| - 2 < 2\,|\mathcal{A}|\,.$$

Now assume that $\mathcal{P}_T \neq \emptyset$. Then $\mathcal{A}$ is the disjoint union of $\mathcal{A}_T$ and $\mathcal{A}[\overline{T}]$ and $\delta_T A = \emptyset$ for each A in $\mathcal{A}$. Hence,

$$\sum_{A \in \mathcal{A}} |\delta_T A| + 2|\mathcal{A}[\overline{T}]| + 2|\mathcal{A}_T| = 2|\mathcal{A}|\,.$$

This proves the corollary. ∎

Corollary 4.3 *Let $\mathcal{P}$ be a partition of V and $(\mathcal{A}, \mathcal{B})$ a bipartition of $\mathcal{P}$. Let T be a tree in $G = (V, E)$. If $|V| \ge 2$ and T is $\mathcal{P}$-connected and has no bridge in $\mathcal{B}$ then*

$$\sum_{A \in \mathcal{A}} |\delta_T A| + |\mathcal{A}[\overline{T}]| + |\mathcal{A}_T| \;\le\; \left(2 - \tfrac{2}{n}\right) |\mathcal{A}|\,, \tag{8}$$

where $n := |V|$.

Proof. Suppose first that $\mathcal{P}_T = \emptyset$, whence also $\mathcal{A}_T = \emptyset$. Then, by Lemma 4.1, we have

$$\begin{aligned}
\sum_{A \in \mathcal{A}} |\delta_T A| + |\mathcal{A}[\overline{T}]| + |\mathcal{A}_T| &= \sum_{A \in \mathcal{A}} |\delta_T A| + |\mathcal{A}[\overline{T}]| \\
&\le \sum_{A \in \mathcal{A}} |\delta_T A| + 2\,|\mathcal{A}[\overline{T}]| \\
&\le 2\,|\mathcal{A}| - 2 \\
&\le 2\,|\mathcal{A}| - 2\,|\mathcal{A}|/n \\
&= \left(2 - \tfrac{2}{n}\right) |\mathcal{A}|\,,
\end{aligned}$$

since $|\mathcal{A}| \le n$. Suppose now that $\mathcal{P}_T \neq \emptyset$. Then $\mathcal{A}$ is the disjoint union of $\mathcal{A}[\overline{T}]$ and $\mathcal{A}_T$ and $\delta_T A = \emptyset$ for each A in $\mathcal{A}$, whence

$$\sum_{A \in \mathcal{A}} |\delta_T A| + |\mathcal{A}[\overline{T}]| + |\mathcal{A}_T| = |\mathcal{A}| \le \left(2 - \tfrac{2}{n}\right) |\mathcal{A}|\,,$$

since $n \ge 2$. ∎

Proof of (i6). Clearly, (i6) holds at the beginning of the first iteration. Assume, by induction, that (i6) holds at the beginning of some non-terminal iteration and let us verify that it holds at the beginning of the next iteration.

Suppose, first, that line 13 is executed. Let $\mathcal{S}' := \mathcal{S} \cup \{S\}$ and let T be some $\mathcal{L}$-connected tree that has no bridge in $\mathcal{S}'$. Since T has no bridge in $\mathcal{S}$, (5) holds. We must show that (5) also holds when y' is substituted for y. Let $\mathcal{P} := \mathcal{L}^*$, $\mathcal{A} := \mathcal{L}^* \setminus \mathcal{S}$, and $\mathcal{B} := \mathcal{L}^* \cap \mathcal{S}$. Lemma 4.1 implies

$$\sum_{L\in\mathcal{A}} |\delta_T L|\,\epsilon + 2\,|\mathcal{A}[\overline{T}]|\,\epsilon + 2\,|\mathcal{A}_T|\,\epsilon \;\le\; 2\,|\mathcal{A}|\,\epsilon\,.$$

The addition of this inequality to (5) produces

$$\sum_{e\in E_T} y'(\mathcal{L}(e)) + 2\,y'(\mathcal{L}[\overline{T}]) + 2\,y'(\mathcal{L}_T) \;\le\; 2\,y'(\mathcal{L})\,,$$

since y' differs from y only in $\mathcal{A}$. Hence, (i6) remains true at the beginning of the next iteration.

Now suppose lines 8–11 are executed. Let $\mathcal{L}' := \mathcal{L} \cup \{L_1, L_2\}$ and let T be an $\mathcal{L}'$-connected tree that has no bridge in $\mathcal{S}$. Since T is $\mathcal{L}$-connected, (5) holds. We must show that (5) also holds when y' and $\mathcal{L}'$ are substituted for y and $\mathcal{L}$ respectively. Let $\mathcal{P} := \mathcal{L}^*$, $\mathcal{A} := \mathcal{L}^* \setminus \mathcal{S}$, and $\mathcal{B} := \mathcal{L}^* \cap \mathcal{S}$. Lemma 4.1 implies $\sum_{L\in\mathcal{A}} |\delta_T L|\,\epsilon + 2\,|\mathcal{A}[\overline{T}]|\,\epsilon + 2\,|\mathcal{A}_T|\,\epsilon \le 2\,|\mathcal{A}|\,\epsilon$, as in the previous case. The addition of this inequality to (5) produces

$$\sum_{e\in E_T} y'(\mathcal{L}'(e)) + 2\,y'(\mathcal{L}'[\overline{T}]) + 2\,y'(\mathcal{L}_T) \;\le\; 2\,y'(\mathcal{L})\,,$$

since $y'_{L_1\cup L_2} = 0$ and y' differs from y only in $\mathcal{A}$. Hence, (i6) remains true at the beginning of the next iteration.

Finally, suppose lines 14–15 are executed. Since neither $\mathcal{L}$ nor $\mathcal{S}$ change in this case, an analysis analogours to that of the first case will show that (5) also holds when y' is substituted for y and therefore (i6) remains true at the beginning of the next iteration. ∎

5 Analysis of the algorithm

Now that we proved invariants (i1) to (i6), we are ready to study the lines 17–25. By virtue of invariant (i3), the tree T returned on line 25 is such that (i3)

$$c(T) \;=\; \sum_{e\in E_T} c_e \;=\; \sum_{e\in E_T} y(\mathcal{L}(e))\,.$$

Next, we must compute a bound on $\pi(\overline{T})$. Suppose first that $\mathcal{M} \ne \emptyset$ and therefore $M \in \mathcal{M}$ and y saturates the complement of M according to invariant (i5). Note that the sets $\bigcup\mathcal{R}$, $\overline{M}$ and (i5) V_T are pairwise disjoint and their union is V. Every element of $\mathcal{R}$ is saturated by y according to invariant (i4). Hence, (i4)

$$\begin{aligned}
\pi(\overline{T}) &= \pi(\overline{M}) + \sum_{R\in\mathcal{R}^*} \pi(R) \\
&= y(\mathcal{L}[\overline{M}]) + y(\mathcal{L}_M) + \sum_{R\in\mathcal{R}^*} y(\mathcal{L}[R]) \\
&= y(\mathcal{L}[\overline{M}]) + \sum_{R\in\mathcal{R}^*} y(\mathcal{L}[R]) + y(\mathcal{L}_M) \\
&\le y(\mathcal{L}[\overline{T}]) + y(\mathcal{L}_M) \\
&\le y(\mathcal{L}[\overline{T}]) + y(\mathcal{L}_T)\,.
\end{aligned}$$

Now suppose $\mathcal{M} = \emptyset$ and therefore M is the only element of $\mathcal{L}^* \setminus \mathcal{S}$. Let $\mathcal{X} := \mathcal{L}^* \cap \mathcal{S}$; of course $\mathcal{X}$ is a partition of $\overline{M}$. Note that $\mathcal{R}^* \cup \mathcal{X} \cup \{V_T\}$ is a partition of V. Every element of $\mathcal{R}$ is saturated
(i4) by y according to invariant (i4). Hence,

$$\begin{aligned}
\pi(\overline{T}) &= \sum_{R\in\mathcal{R}^*} \pi(R) + \sum_{X\in\mathcal{X}^*} \pi(X) \\
&= \sum_{R\in\mathcal{R}^*} y(\mathcal{L}[R]) + \sum_{X\in\mathcal{X}^*} y(\mathcal{L}[X]) \\
&\le y(\mathcal{L}[\overline{T}]) \\
&\le y(\mathcal{L}[\overline{T}]) + y(\mathcal{L}_T)\,.
\end{aligned}$$

Tree T is $\mathcal{L}$-connected. Indeed, for any u and v in $L \cap V_T$, the path from u to v in T is the
(i1) same as in F and uses only vertices of L because of invariant (i1). In addition, the **while** loop on
(i6) lines 21–24 makes sure T has no bridge in $\mathcal{S}$. Hence, invariant (i6) holds for T and therefore

$$c(T) + 2\,\pi(\overline{T}) \;\le\; \sum_{e\in E_T} y(\mathcal{L}(e)) + 2\,y(\mathcal{L}[\overline{T}]) + 2\,y(\mathcal{L}_T) \;\le\; 2\,y(\mathcal{L})\,.$$

(i2) Finally, by virtue of invariant (i2) and Corollary 2.2,

$$c(T) + 2\,\pi(\overline{T}) \;\le\; 2\,\mathrm{opt}(G, c, \pi)\,.$$

Alternatively, we could have shown that $c(T) + \pi(\overline{T}) \le \left(2 - \frac{2}{n}\right) \mathrm{opt}(G, c, \pi)$. This completes the proof of the following theorem:

Theorem 5.1 *Algorithm* GW-Unrooted-Growth *is a* $(2 - \frac{2}{n})$*-approximation for the* PCST *problem.*

The approximation ratio stated in Theorem 5.1 is tight, as the example in Figure 2 shows. The graph in this example is a cycle with n vertices. All edges but one have cost 2. The remaining edge has cost $2 + \rho$. All vertices but the ones incident to this special edge have penalty 1, while the two vertices incident to the special edge have penalty 10. The algorithm increases y_L to 1 for each singleton set L. At this point, all the edges but the special one become tight and are added to the forest F (in some order). After all of them have been added to F, the algorithm stops (phase 2 does nothing in this example) and outputs the tree induced by the dark edges, which has cost $2(n-1)$. The optimum tree, on the other hand, consists only of the special edge (the dark edge depicted in Figure 2(b)). The ratio between the costs of these two solutions approaches $2 - \frac{2}{n}$ as ρ tends to zero.

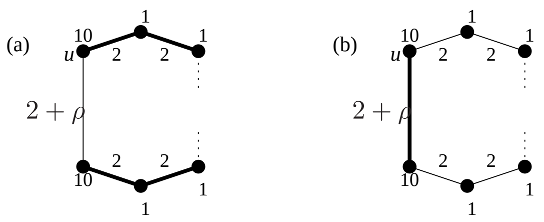

Figure 2: (a) The dark edges indicate the solution produced by the GW-Unrooted-Growth algorithm, a solution of cost $2(n-1)$. (b) The only dark edge indicates the optimum solution, whose cost is $n+\rho$.

6 Running time

Algorithm GW-Unrooted-Growth can be implemented to run in $O(n^2 \log n)$ time. The details of such implementation are analogous to those of the GW algorithm for the rooted PCST (see [4]). There are a few differences, though, that are worth mentioning.

The algorithm does, of course, keep track the numbers y_S, $\pi(S)$, and $\pi(\overline{S})$ for each S in $\mathcal{L}$. In order to compute ϵ and to find tight edges and saturated sets, it must also keep track of the numbers

$$a_S := y(\mathcal{L}[S]) \quad \text{and} \quad b_S := y(\mathcal{L}[\overline{S}]) + y(\mathcal{L}_S)$$

for each S in $\mathcal{L}$, as well as the number $h_f := y(\mathcal{L}(f))$ for each edge f external to $\mathcal{L}^*$. These numbers are updated in each iteration as follows. Between lines 6 and 7, for each S in $\mathcal{L}^* \setminus \mathcal{S}$, we must

add ϵ to a_S,
add $t\epsilon$ to b_L for each L in $\mathcal{L}[S]$,

where $t := |\mathcal{L}^* \setminus \mathcal{S}|$. In addition, for each S in $\mathcal{L}^* \cap \mathcal{S}$ and for each L in $\mathcal{L}[S]$, we must add $t\epsilon$ to b_L. Finally, for each edge f external to $\mathcal{L}^*$, we must add $r\epsilon$ to h_f, where r is the number (0, 1, or 2) of extremes of f in $\mathcal{L}^* \setminus \mathcal{S}$.

Now we must consider the updates required after the decision on lines 6. Between lines 11 and 12, we must

set $a_{L_1 \cup L_2}$ to $a_{L_1} + a_{L_2}$,
set $b_{L_1 \cup L_2}$ to $b_{L_1} - a_{L_2}$ (or, equivalently, to $b_{L_2} - a_{L_1}$).

The overall time required for the management of all these numbers is $O(n)$ per iteration, since $\mathcal{L}$ has $O(n)$ sets. Therefore, it is possible to get an $O(n^2 \log n)$ implementation, as in the GW algorithm.

The ideas proposed by Klein [13] and by Gabow, Goemans and Williamson [7] can also be used, resulting in implementations with running time $O(n\sqrt{m}\log n)$ and $O(n(n+\sqrt{m \log n}))$, respectively, where $m := |E|$. Finally, using the technique of Cole *et al.* [3], one can get a $\big(2 - \frac{2}{n} + \frac{1}{\mathrm{poly}(n)}\big)$-approximation that runs in $O((n+m)\log^2 n)$-time.

7 Previous unrooted growth clustering algorithms

The JMP algorithm seems to be based on the same linear program as the GW algorithm, which differs from the one presented above only in the definition of "y respects π", which consists of (2a)

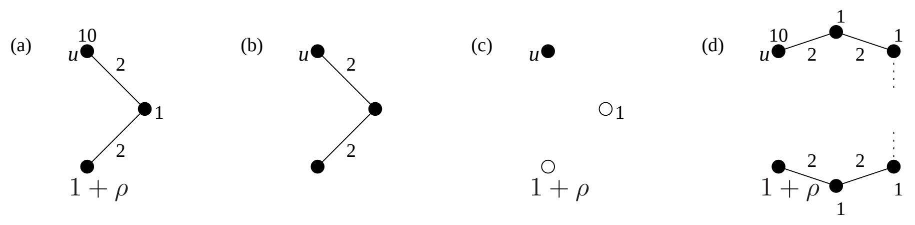

Figure 3: (a) An instance of the PCST. (b) The solution produced by the JMP algorithm when $\rho > 0$. Its cost is 4. (c) The optimum solution, consisting of vertex u alone, has cost $2 + \rho$. (d) A similar instance of arbitrary size consists of a long path.

alone, without (2b). Of course, JMP has no set $\mathcal{M}$ and does not have the case considered in lines 14–15 of our algorithm, Here is the linear program associated with JMP: find vectors x and z that

$$\begin{array}{lrcll} \text{minimize} & \multicolumn{4}{l}{\sum_{e\in E} c_e x_e + \sum_{S\subseteq V} \pi(S) z_S} \\ \text{subject to} & \sum_{e\in\delta L} x_e + \sum_{S\supseteq L} z_S & \geq & 1 & \text{for each } L \subseteq V, \\ & x_e & \geq & 0 & \text{for each } e \in E, \\ & z_S & \geq & 0 & \text{for each } S \subseteq V. \end{array} \tag{9}$$

The dual of this linear program calls for a vector y that

$$\begin{array}{lrcll} \text{maximizes} & \multicolumn{4}{l}{\sum_{L\subseteq V} y_L} \\ \text{subject to} & \sum_{\delta L\ni e} y_L & \leq & c_e & \text{for each } e \in E, \\ & \sum_{L\subseteq S} y_L & \leq & \pi(S) & \text{for each } S \subseteq V, \\ & y_L & \geq & 0 & \text{for each } L \subseteq V. \end{array} \tag{10}$$

(Compare these linear programs with (3) and (4) respectively.) Unfortunately, (9) is not a relaxation of PCST. Hence, its optimum value cannot be used as a lower bound for $\text{opt}(G, c, \pi)$. Algorithm JMP finds a feasible solution of (10) (see Minkoff's thesis [14] for the analysis). The value of this feasible solution might be larger than $\text{opt}(G, c, \pi)$, as the example in Figure 3 shows. Indeed this example shows that the approximation ratio of the JMP algorithm can be arbitrarily close to 2, regardless of the size of the graph. This contradicts Theorem 3.2 in [12].

Minkoff [14] proposed the use of the JMP algorithm for the rooted PCST. (A small adaptation is required, so that the algorithm produces a tree that contains the root vertex.) Unfortunately, the example in Figure 3 with u as root contradicts Theorem 2.6 in [14], which claims that this algorithm is a $(2 - \frac{1}{n})$-approximation.

To our knowledge, there is no published (correct) proof that these two algorithms have any approximation guarantee. We have verified that they are 2-approximations [6]. The proof involves some non-trivial technicalities.

8 Strong pruning

Phase 2 of the GW algorithm, where edges are deleted from the tree produced by phase 1, is sometimes called "pruning phase". This phase can be viewed as an algorithm that solves (exactly or approximately) the PCST problem on trees. When an efficient algorithm is available to solve the problem on trees, it is natural to use it, as it will likely produce solutions better than the usual procedure proposed by Goemans and Williamson.

A dynamic programming approach can solve the PCST on trees in polynomial time. Phase 2 of the GW-Unrooted-Growth algorithm can be replaced by such dynamic programming. This is called "strong pruning". Johnson, Minkoff and Phillips [12] and Minkoff [14] have already suggested this procedure.

With strong pruning, the JMP algorithm is at least as good as without it. But the example shown in Figure 3 does not apply to JMP with strong pruning. The worst example we have for this variant is depicted in Figure 2; it has ratio $2 - \frac{2}{n}$. We conjecture that the JMP algorithm with strong pruning achieves a ratio better than 2. The approximation ratio of our algorithm does not improve by using strong pruning, as the example in Figure 2 shows.

References

[1] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for Prize-Collecting Steiner Tree and TSP. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2009.

[2] F. Chudak, T. Roughgarden, and D.P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming A*, 100(2):411–421, 2004.

[3] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. A faster implementation of the Goemans-Williamson clustering algorithm. In *Symposium on Discrete Algorithms*, pages 17–25, 2001.

[4] P. Feofiloff, C.G. Fernandes, C.E. Ferreira, and J. de Pina. $O(n^2 \log n)$ implementation of an approximation for the Prize-Collecting Steiner Tree Problem. Manuscript: http://www.ime.usp.br/~pf/papers/low-level-pf.pdf, 2002.

[5] P. Feofiloff, C.G. Fernandes, C.E. Ferreira, and J.C. de Pina. Primal-dual approximation algorithms for the Prize-Collecting Steiner Tree Problem. *Information Processing Letters*, 103(5):195–202, 2007.

[6] 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. *CoRR (Computing Research Repository)*, arXiv:1004.1437v2, 2010. Internet: http://arxiv.org/abs/1004.1437v2.

[7] H.N. Gabow, M.X. Goemans, and D.P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming B*, 82(1–2):13–40, 1998.

[8] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 320–309, 1996.

[9] M.X. Goemans. Combining approximation algorithms for prize-collecting TSP. Unpublished manuscript, 2009.

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

[11] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 1997.

[12] D.S. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: theory and practice. In *Symposium on Discrete Algorithms*, pages 760–769, 2000.

[13] P. Klein. A data structure for bicategories, with application to speeding up an approximation algorithm. *Information Processing Letters*, 52(6):303–307, 1994.

[14] M. Minkoff. The prize-collecting Steiner tree problem. Master's thesis, MIT, 2000.