Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 103 (2007) 195–202
Information Processing Letters
www.elsevier.com/locate/ipl

Primal-dual approximation algorithms for the Prize-Collecting Steiner Tree Problem ☆

Paulo Feofiloff, Cristina G. Fernandes *,[1], Carlos E. Ferreira [2], José Coelho de Pina

Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508-090 São Paulo/SP, Brazil

Received 19 September 2006; received in revised form 1 March 2007; accepted 30 March 2007

Available online 13 April 2007

Communicated by K. Iwama

Abstract

The primal-dual scheme has been used to provide approximation algorithms for many problems. Goemans and Williamson gave a $(2-1/(n-1))$-approximation for the Prize-Collecting Steiner Tree Problem that runs in $O(n^3 \log n)$ time—it 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 $(2-2/n)$. 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. Also, algorithms for it have been used for approximating other problems, such as the one of finding a minimum tree spanning k vertices or a k-Steiner tree and the survivable network problem [1,6].

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

Consider a graph $G = (V, E)$, a function c from E into $\mathbf{Q}_{\geqslant}$ (non-negative rationals) and a function π from V into $\mathbf{Q}_{\geqslant}$. For any subset F of E and any subset W of V, let

$$c(F) := \sum_{e \in F} c_e \quad \text{and} \quad \pi(W) := \sum_{w \in W} \pi_w.$$

☆ Research supported in part by PRONEX/CNPq 664107/1997-4 (Brazil).

* Corresponding author.

E-mail addresses: pf@ime.usp.br (P. Feofiloff), cris@ime.usp.br (C.G. Fernandes), cef@ime.usp.br (C.E. Ferreira), coelho@ime.usp.br (J.C. de Pina).

[1] Research supported in part by CNPq Proc. 301174/97-0 (Brazil).

[2] Research supported in part by CNPq Proc. 300752/94-6 (Brazil).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2007.03.012

The Prize-Collecting Steiner Tree Problem (PCST) consists of the following: given G, c, and π, find a tree T in G such that $c(E_T)+\pi(V \setminus V_T)$ is minimum. (We denote by V_H and E_H the vertex and edge sets of a graph H, respectively.) The rooted variant of the problem requires T to contain a given root vertex.

Goemans and Williamson [7,8] used a primal-dual scheme to derive a $(2-1/(n-1))$-approximation for the rooted PCST, where $n := |V|$. Trying all possible choices for the root, they obtained a $(2-1/(n-1))$-approximation for the unrooted PCST. The resulting algorithm runs in time $O(n^3 \log n)$. Johnson et al. [9] proposed a modification of the algorithm that permits running the primal-dual scheme only once, resulting in a running-time of $O(n^2 \log n)$. They claimed the modification, which we refer to as JMP, achieves the same approximation ratio as the original algorithm for the unrooted PCST. Unfortunately, their claim does not hold (as we show here). Cole et al. [2] proposed a faster implementation of Goemans and Williamson's algorithm, which also runs the primal-dual scheme only once, and derived a $(2+1/\text{poly}(n))$-approximation for the unrooted PCST.

This paper contains two results. The main one is a modification of Goemans and Williamson's algorithm for the PCST based on a somewhat different linear program. We show that this new algorithm achieves a ratio of $2-2/n$. It requires only one run of the primal-dual scheme, resulting in a running time of $O(n^2 \log n)$. Also, we present a family of graphs which proves that the given analysis is tight. The second result is an example where the JMP algorithm achieves a ratio of 2. (This shows that their claim on the approximation ratio of their algorithm does not hold.)

Our new algorithm improves slightly the best approximation ratio for the problem. Though the improvement is small, the algorithm seems interesting and might be useful for the design of a better algorithm for PCST. Its behavior is not far from the behavior of the JMP algorithm. It somehow stops before that one does, and, for this, achieves a better ratio. The artifact that makes it stop earlier is subtle and is not obviously polynomially testable.

The paper is organized as follows. The next section introduces some notation and shows some preliminaries. Section 3 has the description of the new algorithm, while its analysis is given in Section 4. Section 5 discusses the JMP algorithm and a variant of it for the rooted PCST. The pruning phase is briefly discussed in Section 6.

2. Notation and preliminaries

The description of the algorithm in the next section will use a notation slightly different from the one in the seminal paper by Goemans and Williamson [7]. With this notation, we found it easier to be sure of the correctness of all the technical details in the analysis.

For any subset X of V, let $\overline{X} := V \setminus X$. For any collection $\mathcal{L}$ of subsets of V and any e in E, let $\mathcal{L}(e) := \{L \in \mathcal{L}\colon e \in \delta_G L\}$, where $\delta_G L$ stands for the set of edges of G with one end in L and the other in $\overline{L}$. Also, $\bigcup \mathcal{L}$ denotes the union of all sets in $\mathcal{L}$. For any function y from $\mathcal{L}$ into $\mathbf{Q}_{\geqslant}$ and any subcollection $\mathcal{M}$ of $\mathcal{L}$, let $y(\mathcal{M}) := \sum_{L\in\mathcal{M}} y(L)$.

We say that y **respects** a function c defined on E (relative to $\mathcal{L}$) if

$$y\big(\mathcal{L}(e)\big) \leqslant c_e \quad \text{for each } e \text{ in } E. \tag{1}$$

An edge e is **tight for** y if equality holds in (1). The inequality in (1) is the usual restriction on edge e: the sum of y_R for all R in $\mathcal{L}$ that e "crosses" does not exceed c_e.

We say y **respects** a function π defined on V (relative to $\mathcal{L}$) if

$$\sum_{S\subseteq L} y_S + \sum_{S\supseteq \overline{L}} y_S \leqslant \pi(L) \quad \text{for each } L \text{ in } \mathcal{L} \tag{2}$$

and

$$\sum_{S\subseteq \overline{L}} y_S + \sum_{S\supseteq L} y_S \leqslant \pi(\overline{L}) \quad \text{for each } L \text{ in } \mathcal{L}. \tag{3}$$

An element L of $\mathcal{L}$ is **tight for** y if equality holds in (2). If equality holds in (3), we say $\overline{L}$ is **tight for** y. The inequality in (2) is slightly different from the usual one for PCST. The usual one says that the sum of y_S for all S in $\mathcal{L}$ contained in L does not exceed the sum of the penalties of all elements in L. In (2), we include in the sum the y_S for supersets S of $\overline{L}$ as well. This has the effect of making the algorithm stop earlier. The inequality in (3) is the same as in (2) for the complement of a set in $\mathcal{L}$.

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 $\mathcal{P}$.

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 maximal elements of a laminar collection $\mathcal{L}$ will be denoted by $\mathcal{L}^*$. So, $\mathcal{L}^*$ is a collection of disjoint subsets of V.

For a graph H and a set S of vertices of H, we denote by $H[S]$ the subgraph of H induced by S. We say

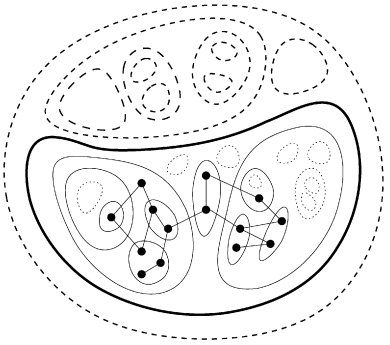

Fig. 1. A connected graph T and a laminar collection of sets whose union contains V_T. In a thicker line, set S as in the proof of Lemma 2.1. The dotted sets form the collection $\mathcal{D}$, the dashed ones together with set S form $\mathcal{C}$ and the remaining sets form $\mathcal{B}$.

a forest F in G is **connected in** a subset L of V if $F[V_F \cap L]$ is connected. For any laminar collection $\mathcal{S}$ of subsets of V, we say a tree T of G **has no bridges in** $\mathcal{S}$ if $|\delta_T S| \neq 1$ (therefore, either $\delta_T S = \emptyset$ or $|\delta_T S| \geqslant 2$) for all S in $\mathcal{S}$.

We denote by $\text{opt}(\text{PCST}(G, c, \pi))$ the minimum value of the expression $c(E_T) + \pi(\overline{V_T})$ when T is a tree in G. The following lower bound serves as motivation for the new algorithm.

Lemma 2.1. *Given a connected subgraph T of G, a laminar collection $\mathcal{L}$ of subsets of V, and a function y from $\mathcal{L}$ into $\mathbf{Q}_{\geqslant}$ that respects c and π, then $y(\mathcal{L}) \leqslant c(E_T) + \pi(\overline{V_T})$.*

Proof. Let S be a minimal set in $\mathcal{L}$ containing V_T. If there is no such set, let $S := V$. Now, consider the partition of $\mathcal{L}$ into the following three sets: $\mathcal{B} := \{L \in \mathcal{L}: \delta_T L \neq \emptyset\}$, $\mathcal{C} := \{L \in \mathcal{L}: L \subseteq \overline{S} \text{ or } L \supseteq S\}$, and $\mathcal{D} := \{L \in \mathcal{L}: L \subseteq S \setminus V_T\}$. (See Fig. 1.) We have that

$$
\begin{aligned}
y(\mathcal{B}) &= \sum_{L\in\mathcal{B}} y_L \leqslant \sum_{L\in\mathcal{B}} |\delta_T L| y_L \\
&= \sum_{e\in E_T} y\big(\mathcal{L}(e)\big) \leqslant \sum_{e\in E_T} c_e = c(E_T), \\
y(\mathcal{C}) &= \sum_{L\in\mathcal{C}} y_L = \sum_{L\subseteq\overline{S}} y_L + \sum_{L\supseteq S} y_L \leqslant \pi(\overline{S}), \\
y(\mathcal{D}) &= \sum_{L\in\mathcal{D}} y_L = \sum_{L\in\mathcal{D}^*} \sum_{X\subseteq L} y_X \\
&\leqslant \sum_{L\in\mathcal{D}^*} \pi(L) \leqslant \pi(S \setminus V_T).
\end{aligned}
$$

The lemma follows from the three inequalities. □

Corollary 2.2. *For any laminar collection $\mathcal{L}$ of subsets of V and any function y from $\mathcal{L}$ into $\mathbf{Q}_{\geqslant}$ that respects c and π, we have that $y(\mathcal{L}) \leqslant \text{opt}(\text{PCST}(G, c, \pi))$.*

The proposed algorithm relies on Corollary 2.2. The above lower bound on $\text{opt}(\text{PCST}(G, c, \pi))$ can also be obtained from the following linear program: find vectors x and z that

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e\in E} c_e x_e + \sum_{L\subseteq V} \pi(L) z_L \\
\text{s.t.} \quad & \sum_{e\in\delta_G S} x_e + \sum_{L\supseteq S} z_L + \sum_{L\supseteq\overline{S}} z_L \geqslant 1 \\
& \qquad \text{for each } S \subseteq V, \\
& x_e \geqslant 0 \quad \text{for each } e \text{ in } E, \\
& z_L \geqslant 0 \quad \text{for each } L \subseteq V.
\end{aligned}
\tag{4}
$$

Given a solution T for $\text{PCST}(G, c, \pi)$, set $x_e := 1$ if $e \in E_T$ and $x_e := 0$ otherwise. Set $z_L := 1$ if $L = V \setminus V_T$ and $z_L := 0$ otherwise. The pair (x, z) is a feasible solution for (4) and its value is $c(E_T) + \pi(V \setminus V_T)$. Thus this program is a relaxation of $\text{PCST}(G, c, \pi)$. Let $\mathcal{R}$ denote the collection of all subsets of V. The dual of this program consists of: find vector y that

$$
\begin{aligned}
\text{maximize} \quad & y(\mathcal{R}) \\
\text{s.t.} \quad & y\big(\mathcal{R}(e)\big) \leqslant c_e \\
& \qquad \text{for each } e \text{ in } E, \\
& \sum_{S\subseteq L} y_S + \sum_{S\supseteq\overline{L}} y_S \leqslant \pi(L) \\
& \qquad \text{for each } L \subseteq V, \\
& y_S \geqslant 0 \quad \text{for each } S \subseteq V.
\end{aligned}
\tag{5}
$$

Linear program (4) is a modification of the original linear program used by Goemans and Williamson for the rooted PCST. The idea behind it is simple. It says that, in the solution given by x and z, for each set S, either there is an edge in $\delta_G S$, or S is contained in a set that pays penalty (a set L with $z_L = 1$), yet or its complement $\overline{S}$ is contained in a set that pays penalty. This difference, though apparently minor, introduces a few interesting questions that might pass unnoticed by a distracted reader. The main one is the following. When growing the values of the y_L variables as in Goemans and Williamson's algorithm, how does one detect that a dual inequality becomes tight? What dual inequalities go tight first? The answers to these questions are not straightforward as in the original Goemans and Williamson's algorithm. Lemma 2.1 and Corollary 2.2 indirectly answer these questions. They guarantee that,

in the algorithm, when looking for tight inequalities, one has only to search among the inequalities for the sets in a laminar collection and for their complements.

Corollary 2.2 is closely related to the traditional argument that any feasible solution for (5) gives a lower bound on $\text{opt}(\text{PCST}(G, c, \pi))$. It is stronger in a sense than this argument though. Indeed, note that (5) has inequalities for all subsets in $\mathcal{R}$, while a y as in Corollary 2.2 is required to satisfy restrictions only to some of these inequalities. It is not straightforward that such a y is a feasible solution for (5). The proof of this fact is very similar to the proof of Lemma 2.1.

3. Unrooted growth clustering algorithm

Our algorithm, which we call GW-UNROOTED-GROWTH, receives G, c, π and returns a tree T in G such that

$$c(E_T) + \pi(\overline{V_T}) \leqslant \left(2 - \frac{2}{n}\right)\text{opt}\big(\text{PCST}(G, c, \pi)\big).$$

We first give a description of the algorithm. It follows the Goemans and Williamson's primal-dual scheme. It consists of two phases. In the first phase a tree is obtained, and in the second phase some edges are possibly removed from this tree to result in the final output.

The first phase works in iterations. Each iteration begins with a spanning forest F and a dual feasible solution y for the linear program (5). We say a component of F is *active* if its vertex set is not a tight set. If there is only one active component, the first phase stops and gives to the second phase the tree induced by F in this component. If there is more than one active component, let $\mathcal{A}$ denote the collection of the vertex sets of active components of F. The algorithm increases uniformly the values of the y variables corresponding to all sets in $\mathcal{A}$, stopping when (at least) one of the three events happens: an external edge, or a set in $\mathcal{A}$, or the complement of a set in $\mathcal{A}$ becomes tight. If the third event happened, the first phase stops and gives to the second phase the tree induced by F in a component of F whose complement became tight. Otherwise, in the first event, a new iteration starts after we added an external tight edge to F. In the second event, a new iteration starts (the only effect is a change in the collection $\mathcal{A}$). If these two previous events happened at the same time, the algorithm chooses arbitrarily any of them.

The second phase does not use the reverse delete strategy as in Goemans and Williamson's pruning algorithm. In fact, it requires that in the first phase we keep track of a collection $\mathcal{S}$ of sets. Roughly, these are the vertex sets of components that in some iteration changed from active to inactive (the second event mentioned above). So, the second phase receives a tree T and this collection $\mathcal{S}$ of sets, and it works in iterations. While there is a set S in $\mathcal{S}$ that has only one edge of T in its boundary, then we remove from T all edges with at least one endpoint in S. The second phase returns the resulting tree.

In the following we provide a detailed version of the algorithm that we need for our analysis. For a graph H and a set S of vertices of H, we denote $H - S$ the graph obtained from H after the removal of all vertices in S. Recall that $H[S]$ denotes the subgraph of H induced by S.

```
GW-UNROOTED-GROWTH (G, c, π)
▷ First phase
 1 F ← (V, ∅)
 2 L ← {{v}: v ∈ V}
 3 S ← ∅        M ← ∅        y ← 0
 4 while |L* \ S| > 1 and M = ∅ do
 5    let ε be the largest number in Q_≥
      such that the function y' respects c and π,
      where y' is defined by y'_L = y_L + ε
      if L ∈ L* \ S, and y'_L = y_L otherwise.
 6    if some edge e external to L* is tight for y'
 7       then let L1 and L2 be the extremes of e in L*.
 8            y'_{L1∪L2} ← 0
 9            F ← F + e
10            L ← L ∪ {L1 ∪ L2}
11    else if some element L of L* \ S is tight for y'
12       then S ← S ∪ {L}
13       else let M in L be such that the set M̄ is tight for y'
14            M ← {M}
15    y ← y'
16 if M ≠ ∅
17    then let M be the only element of M
18    else let M be the only element of L* \ S
19 T0 ← F[M]
▷ Second phase
20 while |δ_{T0} S| = 1 for some S in S do
21    T0 ← T0 − S
22 T ← T0[M] − (⋃ Z ∩ M)
23 return T
```

Note that each iteration of the **while** of the second phase begins a tree T connected in each element of $\mathcal{L}$. At the end, the tree T is connected in each element of $\mathcal{L}$ and has no bridges in $\mathcal{S}$.

As for the first phase of GW-UNROOTED-GROWTH, the following invariants hold in line 4: F is a spanning forest F in 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 $\mathbf{Q}_{\geqslant}$ such that:

(i1) all edges of F are internal to $\mathcal{L}^*$;
(i2) F is connected in each element of $\mathcal{L}$;
(i3) y respects c and π relative to $\mathcal{L}$;
(i4) each edge of F is tight for y;
(i5) each element of $\mathcal{S}$ is tight for y;
(i6) $|\mathcal{M}| \leqslant 1$ and, if $M \in \mathcal{M}$, then $\overline{M}$ is tight for y;
(i7) for any tree T in G, if T is connected in each element of $\mathcal{L}$ and has no bridges in $\mathcal{S}$, then

$$\sum_{e\in E_T} y\big(\mathcal{L}(e)\big) + \sum_{L\subseteq \overline{V_T}} y_L + \sum_{L\supseteq V_T} y_L \leqslant \left(2-\frac{2}{n}\right) y(\mathcal{L}). \tag{6}$$

The proof of these invariants is done in the next section.

4. Analysis of the algorithm

At the beginning of each iteration of while of lines 4–15, invariants (i1) to (i6) hold trivially. Let us verify that invariant (i7) holds as well. It is clear that it holds at the beginning of the first iteration, because $y_L = 0$ for all L in $\mathcal{L}$. Now assume that invariant (i7) holds at the beginning of an iteration where lines 7–10 are executed. Let T be a tree in G, connected in each element of $\mathcal{L}' := \mathcal{L} \cup \{L_1 \cup L_2\}$, with no bridges in $\mathcal{S}$. We must show that (6) holds with $\mathcal{L}'$ and y' in the roles of $\mathcal{L}$ and y. Since $y'_{L_1\cup L_2} = 0$, this is equivalent to

$$\sum_{e\in E_T} y'\big(\mathcal{L}(e)\big) + \sum_{L\subseteq \overline{V_T}} y'_L + \sum_{L\supseteq V_T} y'_L \leqslant \left(2-\frac{2}{n}\right) y'(\mathcal{L}). \tag{7}$$

If $\varepsilon = 0$ then (7) is true because it is identical to (6). Now suppose $\varepsilon > 0$ and let $\mathcal{A} := \mathcal{L}^* \setminus \mathcal{S}$. Since y' differs from y only in $\mathcal{A}$, this is equivalent to

$$\sum_{e\in E_T} \big|\mathcal{A}(e)\big| + \big|\big\{L\in\mathcal{A}\colon\ L\subseteq \overline{V_T}\big\}\big| + \big|\big\{L\in\mathcal{A}\colon\ L\supseteq V_T\big\}\big| \leqslant \left(2-\frac{2}{n}\right)|\mathcal{A}|. \tag{8}$$

Let $\mathcal{N} := \{L \in \mathcal{L}^*\colon\ \delta_T L = \emptyset\}$. Since T is connected, $\mathcal{N} \cap \mathcal{A} = \{L \in \mathcal{A}\colon\ L \subseteq \overline{V_T}\} \cup \{L \in \mathcal{A}\colon\ L \supseteq V_T\}$. As $\sum_{e\in E_T} |\mathcal{A}(e)| = \sum_{L\in\mathcal{A}} |\delta_T L|$ and $(2 - 2/n)|\mathcal{A}| \geqslant 2|\mathcal{A}| - 2$, the inequality (8) will follow from

$$\sum_{L\in\mathcal{A}} |\delta_T L| + |\mathcal{N}\cap\mathcal{A}| \leqslant 2|\mathcal{A}| - 2. \tag{9}$$

If $\mathcal{A} \subseteq \mathcal{N}$, then (9) holds because $\sum_{L\in\mathcal{A}} |\delta_T L| + |\mathcal{N} \cap \mathcal{A}| = |\mathcal{A}| \leqslant 2|\mathcal{A}| - 2$, since $|\mathcal{A}| > 1$ inside the while of lines 4–15. Now assume $\mathcal{A} \not\subseteq \mathcal{N}$ and consider the graph $H := (\mathcal{L}^*, E')$, where E' is the set of edges of T external to $\mathcal{L}^*$ and each element of E' is incident to its two extremes in $\mathcal{L}^*$. Since T is connected in each element of $\mathcal{L}$, this graph is a forest. It has one nontrivial component and $|\mathcal{N}|$ trivial components. Hence, $|E'| = |\mathcal{L}^*| - 1 - |\mathcal{N}|$ and

$$\begin{aligned}
\sum_{L\in\mathcal{A}} |\delta_T L| &= \sum_{L\in\mathcal{L}^*} |\delta_T L| - \sum_{L\in\mathcal{L}^*\cap\mathcal{S}} |\delta_T L| \\
&= 2|E'| - \sum_{L\in\mathcal{L}^*\cap\mathcal{S}} |\delta_T L| \\
&= 2|\mathcal{L}^*| - 2 - 2|\mathcal{N}| - \sum_{L\in\mathcal{L}^*\cap\mathcal{S}} |\delta_T L| \\
&\leqslant 2|\mathcal{L}^*| - 2 - 2|\mathcal{N}| - 2\big|(\mathcal{L}^*\cap\mathcal{S})\setminus\mathcal{N}\big| \qquad (10)\\
&= 2|\mathcal{L}^*| - 2 - 2|\mathcal{N}| - 2|\mathcal{L}^*\cap\mathcal{S}| + 2|\mathcal{N}\cap\mathcal{S}| \\
&= 2|\mathcal{L}^*\setminus\mathcal{S}| - 2 - 2|\mathcal{N}\setminus\mathcal{S}| \\
&\leqslant 2|\mathcal{A}| - 2 - |\mathcal{N}\cap\mathcal{A}|,
\end{aligned}$$

where (10) holds because T has no bridges in $\mathcal{S}$. We have thus shown that invariant (i7) remains valid after the execution of lines 10 and 15. The very same proof applies in the case where line 12 is executed and in the case where lines 13–14 are executed.

Having proved invariants (i1) to (i7), we are ready to analyze the finalization of the algorithm (lines 16–23). By virtue of invariant (i4), the tree T produced by the algorithm is such that

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

If $\mathcal{M} \neq \emptyset$, let $\mathcal{X} := \{\overline{M}\} \cup \mathcal{Z}^*$; else, let $\mathcal{X} := (\mathcal{L}^* \cap \mathcal{S}) \cup \mathcal{Z}^*$. In either case, $\mathcal{X}$ is a collection of disjoint sets. Every element of $\mathcal{X}$ is tight for y, according to invariants (i5) and (i6). Hence,

$$\begin{aligned}
\pi(\overline{V_T}) &= \sum_{X\in\mathcal{X}} \pi(X) \\
&= \sum_{X\in\mathcal{X}} \left(\sum_{L\subseteq X} y_L + \sum_{L\supseteq \overline{X}} y_L\right) \\
&= \sum_{X\in\mathcal{X}} \sum_{L\subseteq X} y_L + \sum_{X\in\mathcal{X}} \sum_{L\supseteq \overline{X}} y_L \\
&\leqslant \sum_{L\subseteq \overline{V_T}} y_L + \sum_{X\in\mathcal{X}} \sum_{L\supseteq \overline{X}} y_L \qquad (11)\\
&\leqslant \sum_{L\subseteq \overline{V_T}} y_L + \sum_{L\supseteq V_T} y_L\,. \qquad (12)
\end{aligned}$$

Inequality (11) holds since every element of $\mathcal{X}$ is disjoint from V_T. In order to explain inequality (12), we

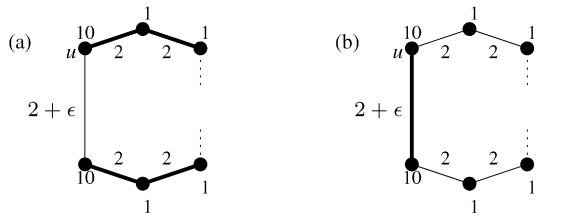

Fig. 2. (a) The dark edges indicate the solution produced by the GW-UNROOTED-GROWTH algorithm, a solution of cost $2(n-1)$. (b) The only dark edge indicates the optimal solution, whose cost is $n+\varepsilon$.

reason as follows. First, observe that $V_T \subseteq \overline{X}$ and therefore $\{L \in \mathcal{L}\colon L \supseteq \overline{X}\} \subseteq \{L \in \mathcal{L}\colon L \supseteq T\}$ for every X in $\mathcal{X}$. Next, $\{L \in \mathcal{L}\colon L \supseteq \overline{X}\} \cap \{L \in \mathcal{L}\colon L \supseteq \overline{X'}\} = \emptyset$ for any two different elements X and X' of $\mathcal{X}$, since $\mathcal{X}$ is a collection of disjoint sets.

The tree T is connected in each element L of $\mathcal{L}$. Indeed, for any u and v in $L \cap V_T$, the path from u to v in T is the same as in F and uses only vertices of L because of invariant (i2). In addition, the while of lines 20–21 makes sure T has no bridges in $\mathcal{S}$. Hence, invariant (i7) holds for T and therefore

$$\begin{aligned} c(E_T) + \pi(\overline{V_T}) &\leqslant \sum_{e \in E_T} y\big(\mathcal{L}(e)\big) + \sum_{L \subseteq \overline{V_T}} y_L + \sum_{L \supseteq V_T} y_L \\ &\leqslant \left(2 - \frac{2}{n}\right) y(\mathcal{L}). \end{aligned}$$

Finally, by virtue of invariant (i3) and Corollary 2.2,

$$c(E_T) + \pi(\overline{V_T}) \leqslant \left(2 - \frac{2}{n}\right) \mathrm{opt}\big(\mathrm{PCST}(G, c, \pi)\big).$$

(In fact, as in Goemans and Williamson's analysis, one can easily prove the stronger statement $c(E_T) + 2\pi(\overline{V_T}) \leqslant (2 - 2/n)\mathrm{opt}(\mathrm{PCST}(G, c, \pi))$.) This completes the proof of the following theorem:

Theorem 4.1. *Algorithm* GW-UNROOTED-GROWTH *is a* $(2 - 2/n)$*-approximation for* $\mathrm{PCST}(G, c, \pi)$.

The approximation ratio stated in Theorem 4.1 is tight, as the example in Fig. 2 shows. The graph in this example is a circuit with n vertices. All edges but one have cost 2. The remaining edge has cost $2 + \varepsilon$. All vertices but the ones incident to this special edge have penalty 1. 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 (the dark edges in Fig. 2(a)) become tight and enter the forest F one by one (in an arbitrary order). When all of them enter F, the algorithm stops (the second phase does nothing in this example) and outputs the tree induced by the dark edges, which has cost $2(n-1)$. The optimal tree, on the other hand, consists only of the special edge (the dark edge depicted in Fig. 2(b)). The ratio between the costs of these two solutions approaches $2 - 2/n$ as ε tends to zero.

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 Goemans and Williamson's algorithm for the rooted PCST. There are a few differences, though, that are worth mentioning. One should carry, for each set L in $\mathcal{L}$, two values that we call $\Delta_1(L)$ and $\Delta_2(L)$, defined as

$$\Delta_1(L) := \Pi(L) - \sum_{S \subseteq L} y_S - \sum_{S \supseteq \overline{L}} y_S$$

and

$$\Delta_2(L) := \Pi(\overline{L}) - \sum_{S \subseteq \overline{L}} y_S - \sum_{S \supseteq L} y_S.$$

In other words, each set L in $\mathcal{L}$ keeps the current slack in inequalities (2) and (3). During the algorithm, one has to keep such values updated. In the algorithm, in each iteration, in lines 7–10, one has to decrease $\Delta_1(L)$ by ε, for every maximal active set L in $\mathcal{L}$, and one has to decrease $\Delta_2(L)$ by $t\varepsilon$, where t is the number of maximal active sets in the iteration, for every L in $\mathcal{L}$. Additionally, one has to set $\Delta_1(L_1 \cup L_2)$ and $\Delta_2(L_1 \cup L_2)$. Roughly, $\Delta_1(L_1 \cup L_2) := \Delta_1(L_1) + \Delta_1(L_2)$ and $\Delta_2(L_1 \cup L_2) := (\Delta_2(L_1) + \Delta_2(L_2) - \Delta_1(L_1) - \Delta_1(L_2))/2$. A few adjustments are needed in $\Delta_1(L_1 \cup L_2)$ when the current forest has few (two or three) components. The overall time required for these calculations is $O(n)$ per iteration, since $\mathcal{L}$ has $O(n)$ sets. Therefore, it is indeed possible to get an $O(n^2 \log n)$ implementation, as in the original Goemans and Williamson's algorithm. (For more details, see [3].)

Also, the ideas proposed by Klein [10] and by Gabow et al. [5] can 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. [2], one can get a $(2 - 2/n + 1/\mathrm{poly}(n))$-approximation that runs in $O((n+m)\log^2 n)$-time.

5. Previous unrooted growth clustering algorithms

The JMP algorithm seems to be based on the same LP as Goemans and Williamson's algorithm, which differs from the one presented above only at the definition

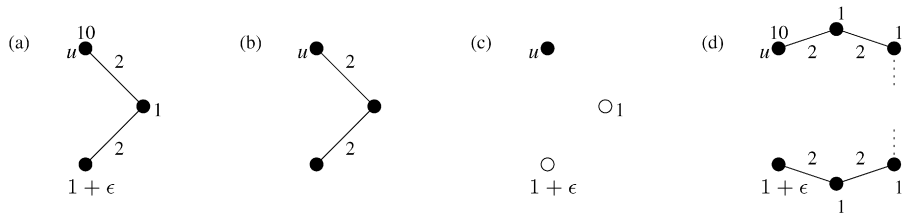

Fig. 3. (a) An instance of the PCST. (b) The solution produced by the JMP algorithm when $\varepsilon > 0$. Its cost is 4. (c) The optimal solution, consisting of only vertex u, has cost $2 + \varepsilon$. (d) A similar instance of arbitrary size.

of "y respects π". In this case, (2) and (3) are replaced by

$$\sum_{S \subseteq L} y_S \leqslant \pi(L) \quad \text{for each } L \text{ in } \mathcal{L}. \tag{13}$$

Set L is said to be tight if equality holds in (13). Of course, JMP has no set $\mathcal{M}$ and does not have the case considered in lines 13–14, since either an edge external to $\mathcal{L}^*$ or a set in $\mathcal{L}^* \setminus \mathcal{S}$ becomes tight for y'. The associate LP would be: find vectors x and z that

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e x_e + \sum_{L \subseteq V} \pi(L) z_L \\ \text{s.t.} \quad & \sum_{e \in \delta_G S} x_e + \sum_{L \supseteq S} z_L \geqslant 1 \\ & \qquad \text{for each } S \subseteq V, \\ & x_e \geqslant 0 \quad \text{for each } e \text{ in } E, \\ & z_L \geqslant 0 \quad \text{for each } L \subseteq V. \end{aligned} \tag{14}$$

(Remember that $\mathcal{R}$ denotes the collection of all subsets of V.) The dual of this linear program is: find vector y that

$$\begin{aligned} \text{maximize} \quad & y(\mathcal{R}) \\ \text{s.t.} \quad & y\big(\mathcal{R}(e)\big) \leqslant c_e \quad \text{for each } e \text{ in } E, \\ & \sum_{S \subseteq L} y_S \leqslant \pi(L) \quad \text{for each } L \subseteq V, \\ & y_S \geqslant 0 \quad \text{for each } S \subseteq V. \end{aligned} \tag{15}$$

Unfortunately, (14) is not a relaxation of PCST, so its optimal value cannot be used as a lower bound for $\text{opt}(\text{PCST}(G, c, \pi))$. Algorithm JMP finds a feasible solution for (15) (see Minkoff's thesis [11] for the analysis). But the value of this feasible solution might be larger than $\text{opt}(\text{PCST}(G, c, \pi))$, as the example in Fig. 3 shows. Indeed this example shows that the approximation ratio of the JMP algorithm can be arbitrarily close to 2, independent of the size of the graph. This contradicts Theorem 3.2 in [9].

Minkoff [11] proposed the use of the JMP algorithm for the rooted PCST. (A small adaptation is required, so that the algorithm produces a tree that always contains the root vertex.) Unfortunately, the example in Fig. 3 with u as root contradicts Theorem 2.6 in [11], which claims that this algorithm is a $(2 - 1/n)$-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 [4]. The analysis that shows this is not straightforward: there are some nontrivial technical details.

6. Strong pruning

The second phase of the algorithm, where edges are deleted from the tree produced in the first phase, is sometimes called pruning phase. This phase can be viewed as an algorithm that solves (exactly or approximately) the original problem in a tree. When an efficient algorithm to solve the problem in a tree exists, it is natural to use it, at least experimentally, as it is likely to 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. The second phase of the GW-UNROOTED-GROWTH algorithm can be replaced by this dynamic programming, which is called "strong pruning". Johnson et al. [9] and Minkoff [11] have already suggested this procedure.

Consider the modified JMP algorithm that uses strong pruning. Its approximation ratio is at least as good as the original one. However, the example shown in Fig. 3 does not apply anymore. The worst example we have for it is depicted in Fig. 2 and the ratio achieved is $2 - 2/n$. 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 Fig. 2 shows.

References

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

[2] R. Cole, R. Hariharan, M. Lewenstein, E. Porat, A faster implementation of the Goemans–Williamson clustering algorithm, in: Symposium on Discrete Algorithms, 2001, pp. 17–25.

[3] P. Feofiloff, C.G. Fernandes, C.E. Ferreira, J. de Pina, $O(n^2 \log n)$ implementation of an approximation for the prize-collecting Steiner tree problem. Manuscript available at http://www.ime.usp.br/~cris/publ/implpcst.ps.gz, 2002.

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

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

[6] 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), 1996, pp. 320–309.

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

[8] D.S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, PWS Publishing, 1997.

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

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

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