$O(n^2 \log n)$ implementation of an approximation for the Prize-Collecting Steiner Tree Problem

Paulo Feofiloff * Cristina G. Fernandes *† Carlos E. Ferreira *‡ José Coelho de Pina *

February 2002

Abstract

We give a low-level description of an $O(n^2 \log n)$ implementation of Johnson, Minkoff and Phillips' approximation algorithm for the Prize-Collecting Steiner Tree Problem.

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 which minimizes the sum of its edge costs and the penalties for the vertices left out of the tree. Johnson, Minkoff and Phillips [2] presented a 2-approximation for the this problem based on the primal-dual scheme. In this manuscript, we describe in details an $O(n^2 \log n)$ implementation of this algorithm.

We adopt the notation used [1], which is summarized below. We start with a formal definition of the problem. Consider a graph $G = (V, E)$, a function c from E into $\mathbb{Q}_{\geq}$ (non-negative rationals) and a function π from V into $\mathbb{Q}_{\geq}$. For any subset F of E and any subset W of V, let $c(F) := \sum_{e \in F} c_e$ and $\pi(W) := \sum_{w \in W} \pi_w$. The **Prize-Collecting Steiner Tree Problem** (PCST) consists of the following: given G, c, and π, find a tree T in G such that

$$c(E_T) + \pi(V \setminus V_T) \text{ is minimum.}$$

(V_H and E_H denote the vertex and edge sets of a graph H.)

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. Let $\bigcup \mathcal{L}$ denote the union of all sets in $\mathcal{L}$.

*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).

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

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

For any collection $\mathcal{L}$ of subsets of V and any subset X of V, let $\overline{X} := V \setminus X$, $\mathcal{L}^X := \{L \in \mathcal{L} : L \subseteq X\}$ and $\mathcal{L}_X := \{L \in \mathcal{L} : L \supseteq X\}$. When $X = \{v\}$, we write $\mathcal{L}^v$ and $\mathcal{L}_v$ instead, and when $X = V_T$ or $X = \overline{V_T}$, we write T or $\overline{T}$ instead. For any e in E, let $\mathcal{L}(e) := \{L \in \mathcal{L} : e \in \delta_G L\}$, where $\delta_G L$ stands for the set of edges of G with one end in L and the other in $\overline{L}$. For any function y from $\mathcal{L}$ into $\mathbb{Q}_\geq$ 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

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

An edge e is **tight for** y if equality holds in (1).

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

$$y(\mathcal{L}^L) \quad \leq \quad \pi(L) \qquad \text{for each } L \text{ in } \mathcal{L}. \tag{2}$$

(3)

2 Johnson, Minkoff and Phillips' algorithm

In its high-level description below, we refer to an algorithm Pruning whose high-level description we omit. It corresponds to the second phase of the primal-dual scheme, where edges are deleted from the tree produced in the first phase.

Johnson, Minkoff and Phillips' algorithm receives G, c, π and returns a tree T in G such that $c(E_T) + \pi(\overline{V_T}) \leq 2\,\mathrm{opt}(\textsc{pcst}(G, c, \pi))$. Each iteration starts with a spanning forest F in G, a laminar collection $\mathcal{L}$ of subsets of V with $\bigcup \mathcal{L} = V$, a subcollection $\mathcal{S}$ of $\mathcal{L}$, and a function y from $\mathcal{L}$ into $\mathbb{Q}_\geq$. The first iteration starts with $F = (V, \emptyset)$, $\mathcal{L} = \{\{v\} : v \in V\}$, $\mathcal{S} = \emptyset$, and $y = 0$. Each iteration consists of the following:

Case 1: $|\mathcal{L}^* \setminus \mathcal{S}| > 1$.

Let ε be the largest number in $\mathbb{Q}_\geq$ such that the function y' defined by

$$y'_L \quad = \quad \begin{cases} y_L + \varepsilon\,, & \text{if } L \in \mathcal{L}^* \setminus \mathcal{S} \\ y_L\,, & \text{otherwise} \end{cases}$$

respects c and π.

Subcase 1A: some edge e external to $\mathcal{L}^*$ is tight for y'.

Let L_1 and L_2 be the extremes of e in $\mathcal{L}^*$. Set $y'_{L_1 \cup L_2} := 0$ and start a new iteration with $F + e$, $\mathcal{L} \cup \{L_1 \cup L_2\}$, $\mathcal{S}$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, y respectively.

Subcase 1B: some element L of $\mathcal{L}^* \setminus \mathcal{S}$ is tight for y'.

Start a new iteration with F, $\mathcal{L}$, $\mathcal{S} \cup \{L\}$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, y respectively.

Case 2: $|\mathcal{L}^* \setminus \mathcal{S}| = 1$.

Let M be the only element of $\mathcal{L}^* \setminus \mathcal{S}$. Call subalgorithm Pruning with arguments $F \cap M$, $\mathcal{L}^M$, and $\mathcal{S}^M$. The subalgorithm returns a subcollection $\mathcal{Z}$ of $\mathcal{S}^M$. Return $T := (F \cap M) - \bigcup \mathcal{Z}$ and stop.

3 Data structures and basic functions

Here is the list of variables and functions used by the algorithm:

1. $L_1, \ldots, L_N$ are nonempty subsets of V_G such that $L_1 \cup \cdots \cup L_N = V_G$ and, for each pair $i < j$, either $L_i \subset L_j$ or $L_i \cap L_j = \emptyset$, whence $N < 2n$, where $n := |V_G|$. Each L_i is represented by a bit vector as well as by a linked list. (In the high-level version of the algorithm given in [1], $\{L_1, \ldots, L_N\}$ is denoted by $\mathcal{L}$.)

2. A subset F of E_G, represented as a doubly-linked list (a bit vector would be too long). Since (V_G, F) is a forest, $|F| < n$.

3. A bit vector $\mu[1\,..\,N]$ such that $\mu[i] = 1$ iff L_i is a maximal element of $\{L_1, \ldots, L_N\}$. (In the high-level version of the algorithm, this set of maximal elements is denoted by $\mathcal{L}^*$.)

4. An array d indexed by V_G with values in $\mathbb{Q}_{\geq}$. (In terms of the high-level notation, $d[v] := \mathcal{L}_v \equiv \sum_{L \in \mathcal{L} : v \in L} y_L$ for each vertex v.)

5. A function ResidualCost that takes edges into $\mathbb{Q}_{\geq}$: upon receiving an edge uv, the function returns the number $c_{uv} - d[u] - d[v]$. Of course this can be implemented to run in $O(1)$ time. (We do not treat ResidualCost as an array because we cannot afford to update ResidualCost every time d changes.)

6. An array $\Delta[1\,..\,N]$ with values in $\mathbb{Q}_{\geq}$.[1] (In terms of the high-level notation, $\Delta[i] = \sum_{v \in L_i} \pi[v] - \sum_{S \subseteq L_i} y_S$.)

7. A bit vector $\lambda[1\,..\,N]$ such that if $\lambda[i] = 0$ then $\Delta[i] = 0$. We say that L_i is *active* iff $\lambda[i] = 1$. (In terms of the high-level notation, $\lambda[i] = 0$ iff $L_i \in \mathcal{S}$.)

8. A variable *mxActive* records the cardinality of the set $\{i : 1 \leq i \leq N, \mu[i] = 1, \lambda[i] = 1\}$.

9. An array $A[1\,..\,N, 1\,..\,N]$ whose elements are sets of at most one edge each. More specifically, for $i \neq j$ such that $\mu(i) = \mu(j) = 1$,

   if $\delta(L_i) \cap \delta(L_j) = \emptyset$ then $A[i, j] = A[j, i] = \emptyset$;
   otherwise, $A[i, j] = A[j, i] = \{uv\}$ where uv is an element of $\delta(L_i) \cap \delta(L_j)$ that minimizes ResidualCost(uv).

10. A function Key defined on $\{1, \ldots, N\} \times \{1, \ldots, N\}$ as follows: if $A[i, j] = \emptyset$ then Key$(i, j) = \infty$; else Key$(i, j) =$ ResidualCost(uv), where uv is the unique edge in $A[i, j]$. Of course this function can be implemented to run in $O(1)$ time.

11. For each i such that $\mu[i] = 1$, there are two subsets of $\{1, \ldots, N\}$ denoted by $H_0[i]$ and $H_1[i]$. For each h, the set $H_h[i]$ consists of all $j \neq i$ such that

$$\mu[j] = 1,\ \lambda[j] = h,\ A[i, j] \neq \emptyset\,.$$

   Each set $H_h[i]$ is organized as a min-heap, the key of each element j being Key(i, j).[2] Hence, the first element of $H_h[i]$ minimizes Key$(i, *)$.

[1] Johnson, Minkoff and Phillips say this is the "surplus" of L_i.
[2] Johnson, Minkoff and Phillips say that the key of j is the "deficit" of the only edge in $A[i, j]$.

12. For $h \in \{0, 1\}$, we assume that we can decide in time $O(1)$ whether or not a statement like "$p \in H_h[i]$" is true or false. Moreover, if the statement is true, we assume that the deletion of p from $H_h[i]$ can de carried out in $O(\log n)$ time. (This is easy to implement: for each i, each h, and each p in $\{1, \ldots, N\}$, maintain the location of p in $H_h[i]$.)

4 Main functions

The core of the algorithm is given by the next functions.

PCST-LOW-LEVEL (G, c, π)
INICIALIZATION()
$N \leftarrow \mathit{mxActive} \leftarrow n$
while $\mathit{mxActive} > 1$ do $\triangleright$ at most $2n$ iterations
    ONEITERATION()
$(X, F) \leftarrow$ PRUNING()
return X and F

The number of iterations is $\leq 2n$ because the sum $2 \times \mathit{mxActive} + \mathit{mxInactive}$, where $\mathit{mxInactive}$ is the cardinality of $\{i : 1 \leq i \leq N, \mu[i] = 1, \lambda[i] = 0\}$, starts at $2n$ and strictly decreases with each iteration.

INICIALIZATION()
$n \leftarrow |V_G|$
$i \leftarrow 0$
for each v in V_G do
    $d[v] \leftarrow 0$
    $i \leftarrow i + 1$
    $L_i \leftarrow \{v\}$
    $o[v] \leftarrow i$
    $\mu[i] \leftarrow \lambda[i] \leftarrow 1$
    $\Delta[i] \leftarrow \pi_v$
for each i in $\{2, \ldots, n\}$ do
    for each j in $\{1, \ldots, i-1\}$ do
        $A[i, j] \leftarrow \emptyset$
        KEY$(i, j) = \infty$
for each i in $\{2, \ldots, n\}$ do
    for each uv in $\delta(L_i)$ do
        if $o[u] = i$
            then $j \leftarrow o[v]$
            else $j \leftarrow o[u]$
        if KEY$(i, j) >$ RESIDUALCOST(uv)
            then $A[i, j] \leftarrow A[j, i] \leftarrow \{uv\}$

21 $F \leftarrow \emptyset$
22 $H_0[i] \leftarrow \emptyset$
23 for each i in $\{1, \ldots, n\}$ do
24 $\quad H_1[i] \leftarrow \emptyset$
25 $\quad$ for each j in $\{1, \ldots, n\} - \{i\}$ do
26 $\quad\quad$ if $A[i, j] \neq \emptyset$ then $H_1[i] \leftarrow H_1[i] \cup \{j\}$

The total time spent executing lines 14–20 is $O(m) = O(n^2)$. The total time spent building the heap $H_1[i]$ in lines 20–21 is $O(n)$. The total spent by INICIALIZATION is $O(n^2)$.

ONEITERATION() $\quad\triangleright$ each call takes $O(n \log n)$ time
01 $\varepsilon' \leftarrow \varepsilon'' \leftarrow \infty$
02 for each p in $\{1, \ldots, N\}$ such that $\mu[p] = \lambda[p] = 1$ do
03 $\quad$ if $\varepsilon' > \Delta[p]$
04 $\quad\quad$ then $\varepsilon' \leftarrow \Delta[p]$
05 $\quad\quad\quad p' \leftarrow p$
06 $\quad$ if $H_0[p] \neq \emptyset$
07 $\quad\quad$ then let q be the first element of $H_0[p]$
08 $\quad\quad\quad$ if $\varepsilon'' > \text{KEY}(p, q)$
09 $\quad\quad\quad\quad$ then $\varepsilon'' \leftarrow \text{KEY}(p, q)$
10 $\quad\quad\quad\quad\quad p'' \leftarrow p$
11 $\quad\quad\quad\quad\quad q'' \leftarrow q$
12 $\quad$ if $H_1[p] \neq \emptyset$
13 $\quad\quad$ then let q be the first element of $H_1[p]$
14 $\quad\quad\quad$ if $\varepsilon'' > \frac{1}{2}\text{KEY}(p, q)$
15 $\quad\quad\quad\quad$ then $\varepsilon'' \leftarrow \frac{1}{2}\text{KEY}(p, q)$
16 $\quad\quad\quad\quad\quad p'' \leftarrow p$
17 $\quad\quad\quad\quad\quad q'' \leftarrow q$
18 $\varepsilon \leftarrow \min(\varepsilon', \varepsilon'')$
19 for each p in $\{1, \ldots, N\}$ such that $\mu[p] = \lambda[p] = 1$ do
20 $\quad \Delta[p] \leftarrow \Delta[p] - \varepsilon$
21 $\quad$ for each v in L_p do
22 $\quad\quad d[v] \leftarrow d[v] + \varepsilon$
23 $\quad\quad \triangleright$ no need to rebuild heaps H_0 and H_1
24 if $\varepsilon = \varepsilon'$
25 $\quad$ then SUBCASE1B(p') $\quad\triangleright$ takes time $O(n \log n)$
26 $\quad$ else SUBCASE1A(p'', q'') $\quad\triangleright$ takes time $O(n \log n)$

Taken together, all executions of line 22 consume $O(n)$ time. The total spent by ONEITERATION is $O(n \log n)$.

```
SubCase1A(p, q)    ▷ merge L_p and L_q; takes time O(n log n)
    let uv be the unique element of A[p, q]
    F ← F ∪ {uv}
    L_{N+1} ← L_p ∪ L_q
    μ[p] ← μ[q] ← 0    ▷ L_p and L_q are no longer maximal
    μ[N + 1] ← 1    ▷ now L_{N+1} is maximal
    if λ[q] = 1 then mxActive ← mxActive − 1
    Δ[N + 1] ← Δ[p] + Δ[q]
    λ[N + 1] ← 1    ▷ now L_{N+1} is active
    for each i in {1, ..., N} such that μ[i] = 1 do
        if Key(p, i) ≤ Key(q, i)
            then A[N + 1, i] ← A[i, N + 1] ← A[p, i]
            else A[N + 1, i] ← A[i, N + 1] ← A[q, i]
    for each h in {0, 1} do
        H_h[N + 1] ← H_h[p]    ▷ time O(1)
        H_h[N + 1] ← H_h[N + 1] − {q}    ▷ time O(log n)
        for each i in H_h[q] do
            if i ∉ H_h[N + 1]
                then H_h[N + 1] ← H_h[N + 1] ∪ {i}    ▷ time O(log n)
            if i ∈ H_h[N + 1] and Key(N + 1, i) > Key(q, i)
                then Decrease-Key(H_h[N + 1], i, Key(q, i))
    for each i in {1, ..., N} such that μ[i] = 1 do
        H_0[i] ← H_0[i] − {q}    ▷ time O(log n)
        H_1[i] ← H_1[i] − {p, q}    ▷ time O(log n)
        if Key(i, N + 1) < ∞
            then H_1[i] ← H_1[i] ∪ {N + 1}    ▷ time O(log n)
    N ← N + 1

SubCase1B(p)    ▷ deactivate L_p
    λ[p] ← 0
    mxActive ← mxActive − 1
    for each i in {1, ..., N} − {p} do
        if p ∈ H_1[i]
            then H_1[i] ← H_1[i] − {p}    ▷ time O(log n)
                 H_0[i] ← H_0[i] ∪ {p}    ▷ time O(log n)
```

```
Pruning ()    ▷ O(n²) time
for i ← N down to 1 do    ▷ "reverse delete"
    if λ[i] = 0    ▷ L_i is inactive
        then degree ← 0
             for each uv in F do    ▷ O(n) time
                 if |{u,v} ∩ L_i| = 1    ▷ O(1) time
                     then degree ← degree + 1
             ▷ degreee is the cardinality of F ∩ δ(L_i)
             if degree ≤ 1
                 then for each uv in F do
                          if |{u,v} ∩ L_i| ≥ 1
                              then F ← F − {uv}
if F ≠ ∅
    then let X be the set of vertices of G[F]
    else let x be a vertex that maximizes π_x
         X ← {x}
return (X, F)
```

References

[1] P. Feofiloff, C.G. Fernandes, C.E. Ferreira, and J. de Pina. Approximation algorithms for the prize-collecting Steiner tree problem. Available at `http://www.ime.usp.br/~cris/publ/`. Submitted, 2004.

[2] 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.