Discrete Applied Mathematics 118 (2002) 25–42

DISCRETE APPLIED MATHEMATICS

Finding the ℓ-core of a tree

Ronald I. Becker[a,b], Yen I. Chang[a], Isabella Lari[c], Andrea Scozzari[c], Giovanni Storchi[c,*]

[a]*Department of Mathematics, University of Cape Town, Rondebosh 7700, South Africa*
[b]*Technion, Haifa, Israel*
[c]*Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università di Roma "La Sapienza" P.le A. Moro 5, 00185 Roma, Italy*

Received 1 December 1999; received in revised form 1 September 2000; accepted 11 July 2001

Abstract

An ℓ-core of a tree $T=(V,E)$ with $|V|=n$, is a path P with length at most ℓ that is central with respect to the property of minimizing the sum of the distances from the vertices in P to all the vertices of T not in P. The distance between two vertices is the length of the shortest path joining them. In this paper we present efficient algorithms for finding the ℓ-core of a tree. For unweighted trees we present an $O(n\ell)$ time algorithm, while for weighted trees we give a procedure with time complexity of $O(n\log^2 n)$. The algorithms use two different types of recursive principle in their operation. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Core; Facility location; Median problems

1. Introduction

Location theory is concerned with the location of facilities on a given network. These facilities can be seen as single points, located at either a vertex or along an arc of the network, or as complex structures, possibly of a given length, in the form of a path or a tree. The criteria for optimality extensively studied in the literature are:

(a) *minisum* criterion in which the sum of the distances from all the vertices of the network to the facility is minimized;
(b) *minimax* criterion in which the distance from the facility to the farthest vertex is minimized;

where the distance between two vertices is the length of the shortest path between the two points. Hakimi first considered both minimax and minisum optimization problems

* Corresponding author. Tel.: +39-6499-10086; fax: +39-6495-9241.
E-mail addresses: rib@maths.uct.ac.za (R.I. Becker), yeni@maths.uct.ac.za (Y.I. Chang), isabella.lari@uniroma1.it (I. Lari), scozzar@rosd.sta.uniroma1.it (A. Scozzari), giovanni.storchi@uniroma1.it (G. Storchi).

related to the location of one site or of several sites in a network [4,5]. In this work we are interested in problems in which the facilities are path-shaped and the criterion is the minisum one. In recent years there has been a growing interest in this field, since in particular several applications require the location of a path-shaped facility instead of a single point or of a set of points. The location of pipelines, of express lanes in a highway and the design of public transportation routes, can be regarded as the location of path-shaped facilities. Hakimi et al. [3], have shown that these problems are NP-hard on general networks while are polynomially solvable on tree networks. Richey [10] presented a pseudo-polynomial time algorithm for the solution of these problems on series-parallel graphs.

A *core* of a tree is defined to be a path that is optimal with respect to the property of minimizing the sum of the distances from each vertex in the tree to the path. Morgan and Slater [8] developed a linear time algorithm for finding a core of a tree network of equal arc lengths and show how to extend this algorithm when there are different arc lengths. Becker [1] also presented a linear time algorithm for finding a minisum path of a tree. Minieka [7] considered the optimal location of facilities of a specified size that are path-shaped or tree-shaped on a tree network. In his work the facility may include part of an arc, that is, in the case of a path facility, the tips of the path may not be vertices of the tree. Peng and Lo [9] presented a recursive $O(n \log n)$ time algorithm for finding a core of specified length, that is, a path with length "exactly" equal to a specified value ℓ, in unweighted trees. They do not allow the possibility to include partial arcs. It is claimed that their method can be applied to trees with integer lengths, but no details are given and the extension is not immediately obvious.

In this paper we consider the more general problem of finding a core on a tree network, with length "at most" a specified value ℓ. We denote the resulting path by the *ℓ-core* of the tree and we present efficient algorithms for this problem. As in Peng and Lo we consider the case where the tips of the path are vertices of the tree. An ℓ-core of a tree can be trivially obtained in $O(n^3)$ time by enumeration. We first provide an algorithm that solves the problem on unweighted trees in $O(n\ell)$ time. Notice that, in the unweighted case, ℓ is bounded by n. For weighted trees we present a recursive algorithm with time complexity of $O(n \log^2 n)$. Note that, as in Peng and Lo, the latter algorithm applied to the unweighted case has a time complexity of $O(n \log n)$. Hence, we can combine the two algorithms for the unweighted case achieving a total time complexity of $O(n \min\{\ell, \log n\})$.

The methods employed use two different types of recursion. Firstly, the tree is given a root and we use a standard *bottom up* and *top down* level by level visit of the rooted tree. This enables rapid calculation when the quantities that need to be computed for a given vertex are related recursively to corresponding quantities at child vertices. This is the method largely used for the $O(n\ell)$ time algorithm for the unweighted case. Similar approaches are taken in the papers of Becker [1], Kim et al. [6] and Tamir [11]. In [1] the general bottom up and top down procedure is outlined with a number of examples. In [6] the authors present a method with $O(n)$ time complexity to solve the Median Subtree Location Problem on tree networks, consisting of minimizing the sum of the

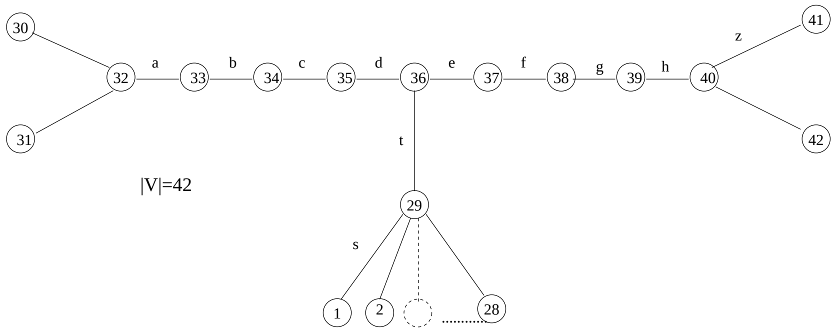


Fig. 1. An example of an ℓ-core.

facility cost plus the sum of the distances of each vertex to the facility. In [11] an improved algorithm for finding a p-median on tree networks is presented. The papers [1,6,11] all use bottom up procedures relating values at one level to values at the present level and also top down procedure to determine further necessary quantities. The problem of finding a path of given length is harder than the above problems for two reasons. Firstly, in [6] a tree facility is sought. Optimal trees in two subtrees of a vertex can be combined to give another tree. In our case, that of a path, combining two such paths may not lead to another path. Secondly, in [6] if we know the optimal facility in a subtree, we can use it unchanged to combine with optimal facilities in a complementary subtree. In our (path) case, the optimal facility in a subtree is a path, but we are not sure what its length is, since it must be combined with another piece of path so as to make the total length ℓ. We require two different types of recursion to cope with this problem. In our weighted case a "central" vertex is computed and the best path of length at most ℓ through it is found. If it is not the ℓ-core, then the ℓ-core must lie entirely in one of the subtrees rooted at the adjacent vertices of the "central" vertex. The algorithm is recursively applied to these subtrees. An appropriate choice of the "central" vertex ensures that the depth of the recursion is $O(\log n)$. The second type of recursion was introduced in Peng and Lo [9].

Our treatment in the weighted case requires the introduction of a data structure which will handle the complexity of combining two paths on each side of a single vertex so as to form a path of length at most ℓ. The complexity is thus reduced from $O(n^3)$ to $O(n\log^2 n)$. Thus this paper improves the complexity in the weighted case to $O(n\log^2 n)$ and in the unweighted case to $O(n\min\{\ell,\log n\})$. Moreover, the problem we consider is a slightly different problem than the one in [9], namely that of finding a path of length at most ℓ in place of a path of length exactly equal to ℓ. The optimal value of our problem can be less than or equal to the optimal value of the latter problem (see Fig. 1). An easy modification of our algorithm will find, in addition, the best path of length exactly equal to ℓ. The paper is organized as follows; in Section 2 we give notation and definitions; Section 3 describes the algorithm for the unweighted ℓ-core. At the end of the section an example is presented. The algorithm for the weighted ℓ-core is described in Section 4. In Section 5 conclusions and further researches are depicted.

2. Notation and definitions

Given a tree $T = (V, E)$, with $|V| = n$, let $w(a) \geqslant 1$ be a weight associated to each edge $a \in E$. Let P be a path in T. The length of P is $L(P) = \sum_{a\in P} w(a)$. Given two vertices u and v, we denote the unique path from u to v as P_{uv}. We define the distance $d(u, v)$ between two vertices u and v of V as the length of P_{uv}. We denote the length of a path P_{uv} by $L(P_{uv})$. Given a path P in T, the distance from P to all the vertices $v \in V$, such that $v \notin P$, is $d(P) = \sum_{v\notin P} d(v, P)$, where $d(v, P)$ is the minimum distance from $v \notin P$ to a vertex of P (see [8]). We call $d(P)$ DISTSUM of P.

An *ℓ-core* of T is a path of length at most ℓ, that minimizes DISTSUM.

Notice that, we consider the case where the path has length at most ℓ, since having fixed ℓ and given a path P of length exactly equal to ℓ that minimizes DISTSUM among all the paths with length exactly equal to ℓ, there could be a path P' having length less than ℓ such that $d(P') < d(P)$.

This is shown in Fig. 1 in the case of an unweighted tree. Suppose that we want to find an ℓ-core with $\ell = 8$. The path P which minimizes DISTSUM among those paths of length exactly equal to 8 is given by the edges $\{a, b, c, d, e, f, g, h\}$ with $d(P) = 61$. While, the ℓ-core P^* of T is given by the edges $\{s, t, e, f, g, h, z\}$, with $d(P^*) = 48$ and length 7.

Given an arbitrary vertex $r \in V$, we may consider r as the root of T, and we denote the resulting rooted tree by T_r. For each edge a we call *tail*(a), the endpoint of a closest to the root and we call *head*(a) the endpoint of a farthest to the root. Each edge a divides T_r into two disjoint subtrees denoted by $T^r_{head(a)}$ and $T^r_{tail(a)}$, having as roots the vertices *head*(a) and *tail*(a) respectively (see Fig. 2). If the root of the tree is understood then we will simply write $T_{head(a)}$ and $T_{tail(a)}$ in place of $T^r_{head(a)}$ and $T^r_{tail(a)}$.

A vertex v is a *leaf* if the number of edges incident to v is equal to 1. Given an edge a we also define *headedges*(a) to be the set of edges incident to *head*(a) not containing a. We indicate with $|T_{head(a)}|$ and $|T_{tail(a)}|$ the cardinality of $T_{head(a)}$ and $T_{tail(a)}$, respectively, where the cardinality of a tree T is the number of vertices of T.

Given a path P_{uv} and a path P_{vw} with edges disjoint from P_{uv}, the *distance saving* of P_{vw} with respect to P_{uv}, is the reduction of DISTSUM obtained by adding P_{vw} to P_{uv} (see [8]), that is:

$$sav(P_{uv}, P_{vw}) = d(P_{uv}) - d(P_{uw}). \tag{1}$$

If the path consists of only one vertex v, we simply write $sav(v, P_{vw})$. In the following *best path* will denote a path satisfying a given property having minimum DISTSUM.

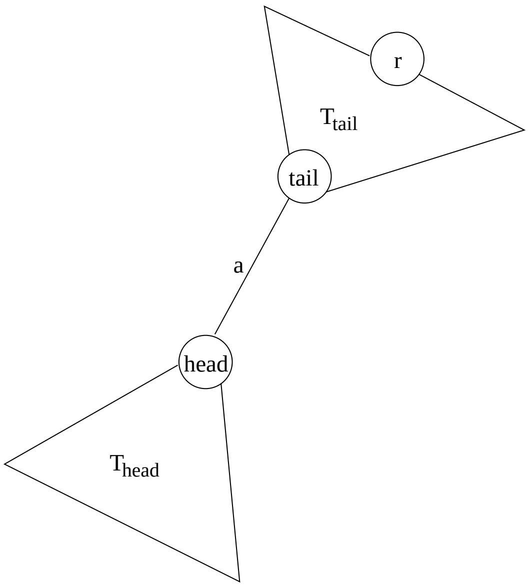


Fig. 2. The two trees separated by an edge a.

3. The algorithm for the unweighted ℓ-core

The basic idea of the algorithm for the unweighted ℓ-core is that for each edge a we find a best path P^* with length at most ℓ, that has a as an extremal edge. In order to find P^*, for each edge a we look for the best path in $T_{head(a)}$, having as endpoint $head(a)$ and length q, with $q=0,1,\ldots,\ell-1$. Similarly we look for the best paths in $T_{tail(a)}$. Then, to find the best paths in $T_{head(a)}$ we use bottom up recursive formulas, while to find the best paths in $T_{tail(a)}$ we need top down recursive formulas. In any event we show that the additional top down phase does not add to the complexity.

We denote by $W^r_{head}(a,q)$ and $W^r_{tail}(a,q)$ the DISTSUM of the best paths in each of the two subtrees of T_r with length exactly equal to $q=0,1,\ldots,\ell-1$. As before for $T^r_{head(a)}$ and $T^r_{tail(a)}$, if the root of the tree is understood we will simply write $W_{head}(a,q)$ and $W_{tail}(a,q)$ in place of $W^r_{head}(a,q)$ and $W^r_{tail}(a,q)$.

In order to compute these weights, we have to consider the maximum distance saving, $Msav_{head}(a,q)$ and $Msav_{tail}(a,q)$, that can be obtained by adding to a a path of length q, with $q=0,1,\ldots,\ell-1$ in $T_{head(a)}$ and in $T_{tail(a)}$, respectively. Note that, changing the mode of the above formulas did not seem to make the formulas any more legible, and we have decided to leave them as they are.

The above quantities can be calculated recursively by using a bottom up and top down level by level visit of the tree as formalized in Becker [1]. In the following we first start by showing how to compute all the *head* recursive functions.

By proceeding up from the edges incident to the leaves of T_r that are in the lowest level until the edges incident to the root are reached, we compute the quantity $|T_{head(a)}|$

as follows:

$$|T_{head(a)}| = \begin{cases} 1 & \text{if } head(a) \text{ is a leaf,} \\ \left\{ \sum_{b \in headedges(a)} |T_{head(b)}| \right\} + 1. \end{cases} \tag{2}$$

To compute the maximum distance saving of a path of length q for each edge a we have:

$$Msav_{head}(a,q) = \begin{cases} 0 & \text{if } q = 0, \\ -\infty & \text{if } head(a) \text{ is a leaf and} \\ & \quad q = 1, 2, \ldots, \ell - 1, \\ \max_{b \in headedges(a)} \{Msav_{head}(b, q-1) + |T_{head(b)}|\} & \text{otherwise.} \end{cases} \tag{3}$$

The following step consists in computing $W_{head}(a,0)\ \forall a$. This weight represents the DISTSUM of the edge a from all the vertices in $T_{head(a)}$.

$$W_{head}(a,0) = \begin{cases} 0 & \text{if } head(a) \text{ is a leaf,} \\ \sum_{b \in headedges(a)} \{W_{head}(b,0) + |T_{head(b)}|\}. \end{cases} \tag{4}$$

In order to find the DISTSUM of the best path $P \cup a$ in $T_{head(a)}$, and such that P has length exactly equal to q we consider:

$$W_{head}(a,q) = W_{head}(a,0) - Msav_{head}(a,q). \tag{5}$$

Proceeding bottom up in T_r it is easy to evaluate the total effort to compute the recursive equations. Since each edge belongs to only one *headedges* set it is considered only once in the right-hand side of the recursive formulas. Hence, the number of required operation is $O(n)$ for (2) and (4) and is $O(n\ell)$ for (3).

Let us consider the problem of computing $|T_{tail(a)}|$, $W_{tail}(a,0)$ and $Msav_{tail}(a,q)$. We will show that for the *tail* formulas it is possible to obtain the same time complexity as the *head* ones.

Having computed $|T_{head(a)}|$ we calculate $|T_{tail(a)}|$ in a constant time by using:

$$|T_{tail(a)}| = n - |T_{head(a)}|. \tag{6}$$

In Fig. 3 there is an example of the calculation of the "*head*" and the "*tail*" cardinalities.

Proceeding top down level by level from the edges incident to the root towards the edges incident to the leaves of the tree, we obtain $W_{tail}(a,0)$ as follows. For each vertex $v \in V$, let S_v be the sum of the distances from the vertices in the subtree of T_r rooted at v to vertex v. We can compute S_v for all v in T_r in $O(n)$ time during a

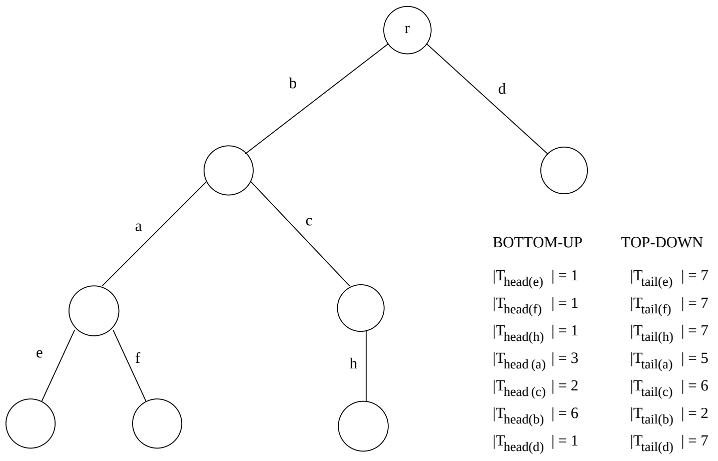


Fig. 3. *head* and *tail* cardinalities of an unweighted tree.

bottom up level by level scan.

$$S_v = \begin{cases} 0 \quad \text{if } v \text{ is a leaf,} \\ \displaystyle\sum_{b \text{ incident to } v} (W_{head}(b,0) + |T_{head(b)}|) \quad \text{otherwise,} \end{cases}$$

where the set of edges incident to v does not contain the parent edge of v. Given the current edge a, we obtain $W_{tail}(a,0)$ as follows:

$$W_{tail}(a,0) = \begin{cases} S_{tail(a)} - W_{head}(a,0) - |T_{head(a)}| \quad \text{if } a \text{ is incident to } r, \\ S_{tail(a)} - W_{head}(a,0) - |T_{head(a)}| + W_{tail}(c,0) + |T_{tail(c)}|, \end{cases} \tag{7}$$

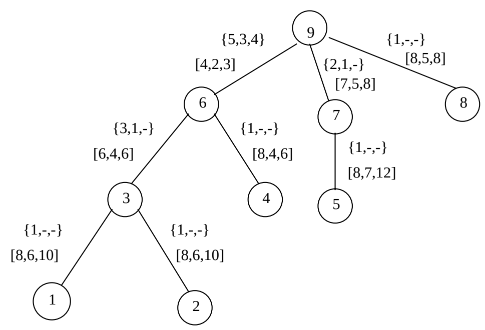
where c is the parent edge of a, that is the edge incident to $tail(a)$ in $T_{tail(a)}$.

Let us consider $Msav_{tail}(a,q)$. Proceeding *bottom up* we can associate to each vertex $v \in V$, three labels, $m_1^q(v)$, $m_2^q(v)$, $\forall q = 1,2,\ldots,\ell - 1$, and $e_1^q(v)$ where $m_1^q(v)$ is the maximum saving obtained by attaching a path of length q in the subtree of T_r rooted at v and starting from v; $e_1^q(v)$ is the edge incident to v contained in the path that gives $m_1^q(v)$; $m_2^q(v)$ is the maximum saving obtained by attaching a path of length q in the subtree of T_r rooted at v, starting from v and which does not contain the edge $e_1^q(v)$.

If v is a leaf, then $m_1^q(v) = m_2^q(v) = -\infty$ $\forall q$. If v has only one edge incident to it then $m_2^q(v) = -\infty$. Notice that, the assignment of the labels to each vertex in the subtree of T_r rooted at v can be performed during a *bottom up* procedure.

Proceeding *top down*, if a is incident to the root r, the maximum saving is:

$$Msav_{tail}(a,q) = \begin{cases} m_1^q(r) \quad \text{if } a \neq e_1^q(r), \\ m_2^q(r) \quad \text{if } a = e_1^q(r). \end{cases} \tag{8}$$



Msav$_{head}$	(1,3)	(2,3)	(3,6)	(4,6)	(6,9)	(7,9)	(5,7)	(8,9)
q=1	-	-	1	-	3	1	-	-
q=2	-	-	-	-	4	-	-	-

Msav$_{tail}$	(1,3)	(2,3)	(3,6)	(4,6)	(6,9)	(7,9)	(5,7)	(8,9)
q=1	6	6	4	4	2	5	7	5
q=2	10	10	6	6	3	8	12	8

BOTTOM-UP

	q=1			q=2		
vertex	m$_1$	m$_2$	e$_1$	m$_1$	m$_2$	e$_1$
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	1	1	(3,1)	-	-	-
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	3	1	(3,6)	4	1	(3,6)
7	1	-	(7,5)	-	-	-
8	-	-	-	-	-	-
9	5	2	(9,6)	8	3	(6,9)

Fig. 4. Maximum savings of length q.

If a is not incident to the root we have:

$$Msav_{tail}(a,q) = \begin{cases} \max\{|T_{tail(b)}| + Msav_{tail}(b,q-1), m_1^q(tail(a))\} & \text{if } a \neq e_1^q(tail(a)), \\ \max\{|T_{tail(b)}| + Msav_{tail}(b,q-1), m_2^q(tail(a))\} & \text{if } a = e_1^q(tail(a)), \end{cases} \tag{9}$$

where b is the parent edge of a.

The time needed for computing $m_1^q(v)$ and $m_2^q(v)$ for all vertices v is O(n). So for all $q = 1,2,\ldots,\ell - 1$ all the labels associated with the vertices can be computed in O($n\ell$) time. Figure Fig. 4 gives an example of the computation of these labels when $q = 1,2$, and shows the table of the maximum savings for each edge.

In Fig. 4, there are two parentheses associated with each edge, where the one with the curly brackets refers to $T_{head(a)}$ and the one with the squared brackets refer to $T_{tail(a)}$ $\forall a$. In particular, the values in each parentheses refer to the cardinality, and the saving of paths of lengths $q = 1,2$, respectively.

In order to find the DISTSUM of the best path $P \cup a$ in $T_{tail(a)}$, and such that P has length exactly equal to q we consider:

$$W_{tail}(a,q) = W_{tail}(a,0) - Msav_{tail}(a,q). \tag{10}$$

Given an edge $a \in E$, to obtain P^*, that is, the best path of length at most ℓ having as extremal edge a in $T_{head(a)}$, we need to compute the weights $\tilde{W}_{head}(a,q)$ that represent the DISTSUM of a best path of length "at most" $q = 0,1,\ldots,\ell - 1$ in $T_{head(a)}$ starting from $head(a)$. We have:

$$\tilde{W}_{head}(a,q)=\begin{cases} W_{head}(a,q) & \text{if } q=0, \\ \min\{W_{head}(a,q),\tilde{W}_{head}(a,q-1)\} & \text{for } q=1,\dots,\ell-1. \end{cases} \tag{11}$$

Given an edge $a\in E$, to obtain P^* in $T_{tail(a)}$, we compute the weights $\tilde{W}_{tail}(a,q)$ that represent the DISTSUM of a best path of length "at most" $q=0,1,\dots,\ell-1$ in $T_{tail(a)}$ starting from $tail(a)$. Then:

$$\tilde{W}_{tail}(a,q)=\begin{cases} W_{tail}(a,q) & \text{if } q=0, \\ \min\{W_{tail}(a,q),\tilde{W}_{tail}(a,q-1)\} & \text{for } q=1,\dots,\ell-1. \end{cases} \tag{12}$$

Next define the DISTSUM $\prod(a)$ of a path P^* of length at most ℓ having as extremal edge a in T as:

$$\prod(a)=\min\{[\tilde{W}_{head}(a,\ell-1)+W_{tail}(a,0)],[\tilde{W}_{tail}(a,\ell-1)+W_{head}(a,0)]\}. \tag{13}$$

The relevance of the above definition is clear since the optimal solution of the problem is achieved by taking the minimum of $\prod(a)$ over all the edges a, where the minimum gives the cost of an ℓ-core. We can keep track of an ℓ-core by using for each a and $q=1,2,\dots,\ell-1$ the following label during a bottom up scan of T_r:

$$y_{head}(a,q)=\text{the edge that gives the maximum distance saving in (3)},$$

$$y_{tail}(a,q)=\text{the edge that gives the maximum distance saving in (8) or (9)}. \tag{14}$$

Then, the major cost of the algorithm for finding an ℓ-core of T consists in computing first all the *head* quantities by the bottom up approach and then, proceeding from the root towards the leaves of the tree, in computing the *tail* quantities.

Theorem 1. *The algorithm for the ℓ-core of an unweighted tree has an overall running time of* $O(n\ell)$.

Proof. The computation of $|T_{head(a)}|$, $|T_{tail(a)}|$, $W_{head}(a,0)$ and $W_{tail}(a,0)$, $\forall a$, can be done in $O(n)$ time as described above. The time needed for finding $Msav_{head}(a,q)$ and $Msav_{tail}(a,q)$ for all a and q is $O(n\ell)$ by using the previous formulas and hence, the time needed for computing $W_{head}(a,q)$, $W_{tail}(a,q)$, $\tilde{W}_{head}(a,q)$ and $\tilde{W}_{tail}(a,q)$ $\forall a$ and $\forall q>0$, is $O(n\ell)$. Finally, the computation of $\prod(a)$ $\forall a\in E$ requires $O(n\ell)$ time. Then, the algorithm runs in $O(n\ell)$ time. □

The correctness of the Procedure follows from the recursive nature of the formulas to be calculated, and it would be tedious to provide further justification. In the following we present an outline of the algorithm.

```
Algorithm: $\ell$-CORE
 begin
  Proceeding bottom up level by level in $T_r$
  for each edge $a \in E$ do
    find $|T_{head(a)}|$   (see (2))
    find $W_{head}(a,0)$   (see (4))
      for each $q = 0,\ldots,\ell-1$ do
        find $Msav_{head}(a,q)$   (see (3))
  for each edge $a \in E$ do
    for each value of $q$ do
      find $W_{head}(a,q)$   (see (5))
      find $\widetilde{W}_{head}(a,q)$   (see (11))
  for each edge $a \in E$ do
    compute $|T_{tail(a)}|$   (see (6))
  Proceeding top down level by level in $T_r$
  for each edge $a \in E$ do
    find $W_{tail}(a,0)$   (see (7))
      for each $q = 0,\ldots,\ell-1$ do
        find $Msav_{tail}(a,q)$   (see (8) or (9))
  for each edge $a \in E$ do
    for each value of $q$ do
      find $W_{tail}(a,q)$   (see (10))
      find $\widetilde{W}_{tail}(a,q)$   (see (12))
  for each edge $a$ do
    compute $\prod(a)$   (see (13))
  Find an edge $a$ that has the minimum $\prod(a)$
  Starting from $a$, build an $\ell$-core   (see (14))
 end
```

3.1. An example

We want to find the ℓ-core of the tree represented in the following figure for $\ell = 1,2,\ldots,6$.

We consider the vertex 2 as the root of the tree. The two labels associated with each edge represent the *head* and the *tail* cardinality, according to the definitions given and to (2) see Fig. 5. Then, we have to compute the *maximum distance saving* referring to (3) for each edges and $\forall q$. The values are stored in Tables 1 and 2, respectively.

From Tables 1 and 2 we can obtain $W_{head}(a,q)$ and $W_{tail}(a,q)$ $\forall a$, (Tables 3 and 4). Finally by referring to (11) and (12) we are able to determine $\widetilde{W}_{head}(a,q)$ and $\widetilde{W}_{tail}(a,q)$. Hence, by using $\prod(a)$ $\forall a$ we can find a core of length ℓ, for $\ell = 1,2,\ldots,6$ (Table 5).

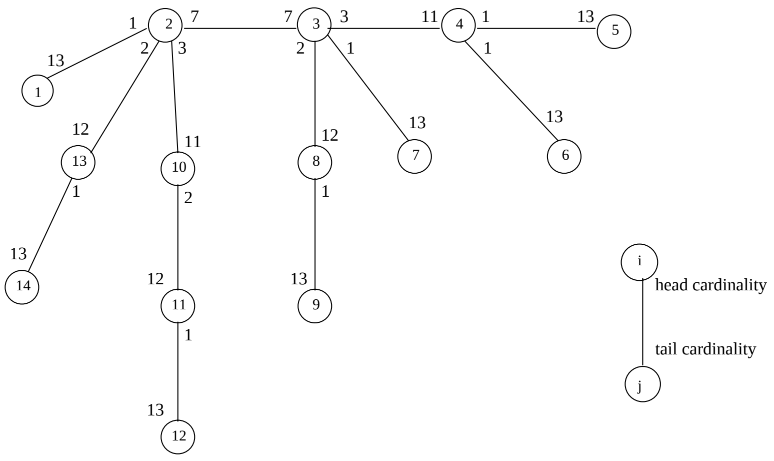


Fig. 5. An example.

Table 1
Msav$_{head}$ of length *q*

11,12	8,9	4,5	4,6	13,14	10,11	3,8	3,7	3,4	1,2	2,13	2,10	2,3	*q*
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1	1	$-\infty$	1	$-\infty$	1	2	3	1
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	3	4	2
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	3
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	4
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	5

Table 2
Msav$_{tail}$ of length *q*

11,12	8,9	4,5	4,6	13,14	10,11	3,8	3,7	3,4	1,2	2,13	2,10	2,3	*q*
12	12	11	11	12	11	7	7	7	7	7	7	3	1
23	19	18	18	19	18	10	10	10	10	10	10	5	2
30	22	21	21	22	21	12	12	12	11	11	11	6	3
33	24	23	23	23	22	13	13	13	$-\infty$	$-\infty$	$-\infty$	$-\infty$	4
34	25	24	24	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	5

Table 3
head weights

11,12	8,9	4,5	4,6	13,14	10,11	3,8	3,7	3,4	1,2	2,13	2,10	2,3	*q*
0	0	0	0	0	1	1	0	2	0	1	3	9	0
∞	∞	∞	∞	∞	0	0	∞	1	∞	0	1	6	1
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	5	2
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	3
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	4
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	5

Table 4
tail weights

11,12	8,9	4,5	4,6	13,14	10,11	3,8	3,7	3,4	1,2	2,13	2,10	2,3	q
43	35	33	33	35	31	23	25	21	25	23	20	10	0
31	23	22	22	23	20	16	18	14	18	16	13	7	1
20	16	15	15	16	13	13	15	11	15	13	10	5	2
13	13	12	12	13	10	11	13	9	14	12	9	4	3
10	11	10	10	12	9	10	12	8	∞	∞	∞	∞	4
9	10	9	9	∞	∞	∞	∞	∞	∞	∞	∞	∞	5

Table 5
Cores of length ℓ

$\ell=1$	(2,3)
$\ell=2$	(2,3)(3,4) or (2,3)(2,10)
$\ell=3$	(2,3)(3,4)(2,10)
$\ell=4$	(2,3)(3,4)(2,10)(10,11)
$\ell=5$	(2,3)(3,4)(2,10)(10,11)(11,12) or (2,3)(3,4)(2,10)(10,11)(4,6)
$\ell=6$	(2,3)(3,4)(2,10)(10,11)(11,12)(4,5) or (2,3)(3,4)(2,10)(10,11)(4,6)(11,12)

4. The algorithm for the weighted ℓ-core

The algorithm given in Section 3 for unweighted trees can be generalized to the case of weighted trees, but the time complexity $O(n\ell)$ may not be good, since ℓ may be larger than n. The aim of this section is to give a more efficient algorithm for the ℓ-core on weighted trees. The task of computing the best path of length at most ℓ is significantly easier if it is known that it passes through a fixed vertex. In this case a "central" vertex is computed and the best path of length at most ℓ through it is found. If it is not the ℓ-core, then the ℓ-core must lie entirely in one of the subtrees rooted at the adjacent vertices of the "central" vertex. The algorithm is recursively applied to these subtrees.

Peng and Lo [9] present a similar recursive algorithm, that gives a core with length exactly equal to ℓ in $O(n\log n)$ time, only in unweighted trees. In their procedure, given a rooted tree, by proceeding top down level by level, they generate the "best" path of length $k=1,\dots,\ell$ from the root to the other vertices of the tree. These paths are obviously already ordered by length, since in unweighted trees the length of a path is the number of edges in the path. Then they combine for each k the path with length k with the path with length $\ell-k$.

In the weighted case, all the paths from the root to the other vertices have in general different lengths. For that reason, one has to maintain all the paths from the root to the other vertices, not only the "best" ones and, moreover, the paths generated proceeding top down level by level are not already ordered by length. Then, if we do not order the paths, the time needed for finding the "best" path containing the root with length at most ℓ is $O(n^2)$, because we have to combine all pairs of paths from the root. If the paths are ordered we can find the "best" path containing the root in $O(n)$ time. Then, in order to reduce the time complexity we first sort the paths in $O(n\log n)$ time.

Given a vertex $v \in V$, we can compute in $O(n)$ time the DISTSUM and the lengths of all the paths having as endpoint v. Consider v as the root of T and, as in Section 3, by a bottom up scan of T_v, we find, $\forall a \in E$, the cardinality of the subtree $T^v_{head(a)}$ and the sum of the distances from $head(a)$ to the vertices in $T^v_{head(a)}$, that is, the weight $W^v_{head}(a,0)$. Given a vertex $u \in V \setminus \{v\}$, let $p(u)$ be its parent. We write $w(p(u),u)$ instead of $w((p(u),u))$, and similarly for other functions of edges. The distance saving $sav(v,P_{vu})$ obtained by adding P_{vu} to v, is given by:

$$sav(v,P_{vu}) = \begin{cases} w(p(u),u)|T_{head(p(u),u)}| & \text{if } p(u)=v, \\ sav(v,P_{vp(u)}) + w(p(u),u)|T_{head(p(u),u)}| & \text{if } p(u)\neq v. \end{cases} \tag{15}$$

Hence we can compute these distance saving for each path P_{vu}, $\forall u \in V \setminus \{v\}$ by a top down scan of T_v. Moreover, as has been computed by a number of authors, for $d(v)$ we have:

$$d(v) = \sum_{a \text{ incident to } v} (W_{head}(a,0) + w(a)|T_{head(a)}|). \tag{16}$$

Then $\forall u$ we can compute the DISTSUM of the path P_{vu}:

$$d(P_{vu}) = d(v) - sav(v,P_{vu}).$$

Finally, by a top down scan of T_v, we can find the lengths of the paths P_{vu} from v to each $u \in V \setminus \{v\}$ as follows:

$$L(P_{vu}) = L(P_{vp(u)}) + w(p(u),u). \tag{17}$$

Hence, for each vertex $v \in V$, we can compute the distance saving, the DISTSUM and the lengths of all the paths having an extreme in v, in $O(n)$ time. Notice that, by applying this procedure for each vertex $v \in V$, we can find the ℓ-core in a weighted tree in $O(n^2)$ time. When $\ell > n$ this time complexity is better than $O(n\ell)$.

The removal of a vertex v in T and all its incidents edges splits the tree into one or more subtrees.

Remark 1. Given a tree $T=(V,E)$ and a vertex $v \in V$ we have two cases:
- the ℓ-core contains v;
- the ℓ-core is fully contained in one of the subtrees obtained by removing v from T.

Definition. Given a weighted tree T, a *central vertex* v of T is the *centroid* of the corresponding unweighted tree of T, that is a vertex that minimizes the maximum of the number of vertices of the subtrees obtained by removing it.

Remark 2. By [5, Lemma 3.1], a central vertex has maximum subtree cardinality less than or equal to $n/2$. By [5, Theorem 3.1], a (unweighted) median of a tree is a central vertex.

By Remark 1 the following recursive algorithm finds the ℓ-core of T.

```
Algorithm SPLIT-TREE:
Input: a weighted tree $T$
Output: an $\ell$-core $P^*$ of $T$ and its DISTSUM $d^*$
  begin
    $d^* := +\infty$
    SUBTREE($T$)
  end

Procedure SUBTREE($T'$):
Input: a subtree $T' = (V', E')$ of $T$ with $n'$ vertices and the best current
DISTSUM $d^*$
Output: if the best path in $T'$ has DISTSUM less than $d^*$, the best path $P^*$ in
$T'$ with length at most $\ell$ and its distsum $d^*$

  begin
    find a central vertex $v$ of $T'$
    BESTPATH($T', v$) {finds a best path $P'$ in $T'$ with length at most $\ell$
    and containing $v$}
    if $d(P') < d^*$ then
      $d^* := d(P')$
      $P^* := P'$
    for each subtree $T^i$ obtained from $T'$ by removing $v$ do
      SUBTREE($T^i$)
  end
```

In procedure BESTPATH(T', v), we consider all the paths in T' having v as an endpoint and length at most ℓ and we sort them in non-increasing order of length. Notice that the number n'' of these paths is at most $n' - 1$. Let P^i, with $i = 1, \dots, n''$, be the ith path in the order. Let $j(i)$ be the maximum index j, such that:

$$L(P^i) + L(P^j) \leqslant \ell. \tag{18}$$

If such an index does not exist, we let $j(i) = 0$. We denote by P^0 the path composed only by the vertex v.

The idea is that we find the best path containing v by combining P^i, for each $i = 0, \dots, n''$, with the best path P^j with $j \leqslant j(i)$. In particular, if $T^1, T^2, \dots, T^h$ are the subtrees obtained by removing v from T', we have to consider only pairs of paths P^i and P^j, $j \leqslant j(i)$, lying in different subtrees. In this case the DISTSUM of a path P obtained from the connection of two paths P^i and P^j is:

$$d(P) = d(v) - sav(v, P^i) - sav(v, P^j). \tag{19}$$

Hence, for each path P^i with $i=0,\dots,n''$, we have to find a path P^j with $j \leqslant j(i)$, lying in a different subtree with respect to that of P^i and with maximum distance saving.

In order to do that, we have to consider, for each $i=1,\dots,n''$, the following quantities:

$t(i) =$ a label that indicates the subtree in which P^i lies;

$Bsav(i) =$ the maximum distance saving of a path P^j with j belonging to $\{0,1,\dots,i\}$;

$Bpath(i)$ =the index of the path with length at most $L(P^i)$ and with maximum distance saving;

$Bsav'(i)$ =the maximum distance saving of a path P^j with j belonging to $\{0,1,\dots,i\}$ and with label different from $t(Bpath(i))$;

$Bpath'(i)$ =the index of the path with length at most $L(P^i)$, with label different from $t(Bpath(i))$ and with maximum distance saving.

For the sake of simplicity for $i=0$ we let $t(0)=0$, $Bsav(0)=0$ and $Bpath(0)=0$.

The labels $t(i)$ for $i=1,\dots,n'-1$ can be obtained when P^i is found, while we can obtain the other previous quantities by the following formulas:

$$Bsav(i) = \max\{sav(v,P^i), Bsav(i-1)\} \quad i=1,\dots,n''. \tag{20}$$

$$Bpath(i) = \begin{cases} i & \text{if } Bsav(i) = sav(v,P^i) \\ Bpath(i-1) & \text{otherwise.} \end{cases} \tag{21}$$

$$Bsav'(i) = \begin{cases} Bsav(i-1) & \text{if } t(Bpath(i-1)) \neq t(Bpath(i)) \\ \max\{sav(v,P^i), & \text{if } t(i) \neq t(Bpath(i)) \text{ and} \\ \quad Bsav'(i-1)\} & \quad t(Bpath(i-1)) = t(Bpath(i)) \\ Bsav'(i-1) & \text{otherwise.} \end{cases} \tag{22}$$

$$Bpath'(i) = \begin{cases} Bpath(i-1) & \text{if } t(Bpath(i-1)) \neq t(Bpath(i)) \\ i & \text{if } t(Bpath(i-1)) = t(Bpath(i)), \\ & t(i) \neq t(Bpath(i)) \\ & \text{and } sav(v,P^i) > Bsav'(i-1) \\ Bpath'(i-1) & \text{otherwise.} \end{cases} \tag{23}$$

To find the best path Q^i containing P^i, we attach P^i to:

$$\begin{array}{ll} P^{Bpath(j(i))} & \text{if } t(i) \neq t(Bpath(j(i))) \\ P^{Bpath'(j(i))} & \text{otherwise.} \end{array} \tag{24}$$

In the following we present an outline of the procedure BESTPATH(T', v).

```
*Procedure*: BESTPATH($T', v$):
 **begin**
  find the DISTSUM $d(v)$ (see (16))
  **for all** vertices $u \neq v$ in $T'$ **do**
    find $sav(v, P_{vu})$ (see (15))
  find $T^1, T^2, \ldots, T^h$, the subtrees obtained from $T'$ by removing $v$
  find all paths $P_{vu}$ for which $L(P_{vu}) \leqslant \ell$ (see (17))

  *sort* all the paths in non-increasing order of length

  Let $P^1, P^2, \ldots, P^{n''}$ be these paths

  **for each** $i = 0, \ldots, n''$ **do**
    let $t(i) = k$ be the label of the subtree $T^k$ to which $P^i$ belongs
  **for each** $i = 0, \ldots, n''$ **do**
    find $j(i)$ (see (18))
    find $Bsav(i)$ (see (20))
    find $Bpath(i)$ (see (21))
  **for each** $i = 1, \ldots, n''$ **do**
    find $Bsav'(i)$ (see (22))
    find $Bpath'(i)$ (see (23))
  **for each** $i = 0, \ldots, n''$ **do**
    find the best path $Q^i$ containing $P^i$ (see (24))
  Choose the path $Q^i$ with $i = 1, \ldots, n''$ which has the minimum DISTSUM
 **end**
```

4.1. The complexity of the algorithm

Let us consider the time complexity of the procedure BESTPATH(T', v). As noted at the beginning of Section 4, we can compute the DISTSUM of a vertex v and the distance saving $sav(v, P_{vu})$ for all the vertices $u \in V' \setminus \{v\}$ in $O(n')$ time. The following remark allows us to calculate the DISTSUM and the distance saving in $O(n')$ time in the subtree T' of T.

Remark 3. Given a tree T and given a vertex z, let us consider T_z. Let $a = (z, u)$ be an edge incident to z and $T' = (V', E')$ be the subtree $T^z_{head(a)}$. Moreover let $d' = W^z_{head}(a, 0)$ be the sum of the distances from all the vertices in T' to u and $n' = |T^z_{head(a)}| = |T^z_u|$ be the number of vertices of T'. Now we consider a vertex v in $T^z_{head(a)}$ as the new root of T and we call T_v the rooted tree so obtained. We have:

$|T^v_{head(a)}| = |T^v_z| = n - n'$ (note that z is $head(a)$ with respect to the new root v),
$W^v_{head}(a, 0) = d(z) - (d' + w(a)n')$.

Hence, having already computed in T_z the cardinalities $|T^z_{head(a)}|$ and the weights $W^z_{head}(a,0)$ for each edge $a \in E$, we can compute the corresponding quantities in T_v, that is $|T^v_{head(a)}|$ and $W^v_{head}(a,0)$ in $O(n')$ time.

Lemma 1. *The solution time of the procedure BESTPATH*(T', v) *is* $O(n' \log n')$ *for any* $v \in V$.

Proof. By Remark 3 we can compute $sav(v, P_{vu})$ for each $u \neq v$ in T' and $d(v)$ in $O(n')$ time, since by (15) and (16) these quantities depend only on $|T^v_{head(a)}|$ and $W^v_{head}(a,0)$. Moreover by using (17), we can compute the lengths $L(P_{vu})$ for all $u \neq v$ in T' also in $O(n')$ time by a top down scan. Sorting the paths P^i can be done in $O(n' \log n')$ time. We can find all the indices $j(i) = 1, \ldots, n''$ in $O(n')$ time since n'' is at most $n' - 1$ and since if $i' > i$ then $j(i') \leqslant j(i)$.

We can find $t(i)$ when $P^i = P_{vu}$ is found by checking which T^j contains u — this adds linear time for a reasonable data structure. The quantities $Bsav(i)$, $Bpath(i)$, $Bsav'(i)$ and $Bpath'(i)$ in the procedure can also be calculated with a time complexity of $O(n')$ by using (20)–(23). □

The time complexity of the Algorithm SPLIT-TREE depends on the depth of the recursion, that is, depends on the cardinalities of the subtrees obtained by removing the central vertex of each current tree T'.

Theorem 2. *The algorithm SPLIT-TREE finds the* ℓ*-core of* T *in* $O(n \log^2 n)$.

Proof. A central vertex v of a tree T can be found in $O(n)$ time, by applying the algorithm for the median of an unweighted tree T presented in [2] (see Remark 2). By Lemma 1 the procedure BESTPATH(T', v) can be implemented in $O(n' \log n')$ time.

Let us now consider the recursive procedure SUBTREE. At a given level of the recursion there are a number k of subproblems to be solved. Let n_i with $i = 1, \ldots, k$ be the number of vertices of the ith subproblem. In order to solve it, we need $O(n_i \log n_i)$ time for finding the best path P' in T'. Since $\sum_{i=1}^{k} (n_i) < n$ we obtain, for each level of the recursion, a time complexity of $O(n \log n)$.

Moreover, the depth of the recursion is $O(\log n)$ since v is a central vertex (the cardinalities of the subtrees obtained by removing a central vertex v are at most $n/2$ by Remark 2). Hence, the overall execution time of the algorithm SPLIT-TREE is $O(n \log^2 n)$. □

Notice that, as in the paper of Peng and Lo, the algorithm SPLIT-TREE finds an ℓ-core in $O(n \log n)$ time when $w(a) = 1 \ \forall a \in E$, i.e. in the unweighted case. Hence, in the unweighted case we have two algorithms which have a combined complexity of $O(n \min\{\ell, \log n\})$ time.

5. Conclusion

In this paper we have presented the problem of finding an *ℓ-core* of a tree network, that is, a path with length *at most* ℓ which minimizes the sum of the distances from it to the other vertices of the tree. Peng and Lo [9] first provided a recursive $O(n \log n)$ time algorithm for finding a path with length *exactly* equal to a specified value ℓ only in unweighted trees. It is claimed that their method can be applied to trees with integer lengths, but no details are given and the extension is not immediately obvious. Our problem can be considered an extension of the latter one, since the optimal value of our problem can be less than or equal to the optimal value in [9]. We have also considered the ℓ-core problem both in unweighted trees and in weighted ones. Then, two algorithms have been provided, one with time complexity of $O(n\ell)$ for unweighted trees and the other for weighted trees which runs in $O(n \log^2 n)$ time. Notice that, as in [9], the latter algorithm applied when the weigths associated to each edge are all equal to 1, has a time complexity of $O(n \log n)$. Hence, we can combine the two procedures for the unweighted case achieving a total time complexity of $O(n \min\{\ell, \log n\})$.

The methods employed use two types of recursion, in an unweighted tree we use a standard bottom up and top down level by level visit of the rooted tree, while in the weighted case the algorithm is based on a divide and conquer technique.

The result presented might provide a basis for the study of other facility shapes such as trees, cycles and for other measures of distances. Ultimately, one would hope to obtain good algorithms for location in general networks.

References

[1] R.I. Becker, Inductive algorithms on finite trees, Quaestiones Math. 13 (1990) 165–181.

[2] A.J. Goldman, Optimum center location in simple networks, Transportation Sci. 5 (1971) 212–221.

[3] S.L. Hakimi, E.F. Schmeichel, M. Labbè, On locating path or tree shaped facilities on network, Networks 23 (1993) 543–555.

[4] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems. I: The p-centers, SIAM J. Appl. Math. 37 (1979) 513–538.

[5] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems. II: The p-medians, SIAM J. Appl. Math. 37 (1979) 539–560.

[6] T.U. Kim, T.J. Lowe, A. Tamir, J.E. Ward, On the location of a tree-shaped facility, Networks 28 (1996) 167–175.

[7] E. Minieka, The optimal location of a path or tree in a tree network, Networks 15 (1985) 309–321.

[8] C.A. Morgan, J.P. Slater, A linear algorithm for a core of a tree, J. Algorithms 1 (1980) 247–258.

[9] S. Peng, W. Lo, Efficient algorithms for finding a core of a tree with a specified length, J. Algorithms 20 (1996) 445–458.

[10] M.B. Richey, Optimal location of a path or tree on a network with cycles, Networks 20 (1990) 391–407.

[11] A. Tamir, An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs, Oper. Res. Lett. 19 (1996) 59–64.