ELSEVIER Discrete Applied Mathematics 87 (1998) 229–243

DISCRETE APPLIED MATHEMATICS

Fully polynomial approximation schemes for locating a tree-shaped facility: A generalization of the knapsack problem

Arie Tamir

Raymond and Beverly Sackler Faculty of Exact Sciences, Department of Statistics and Operations Research, Tel Aviv University, Ramat-Aviv, 69978 Israel

Abstract

Given an n-node tree $T=(V,E)$, we are concerned with the problem of selecting a subtree of a given length which optimizes the sum of weighted distances of the nodes from the selected subtree. This problem is NP-hard for both the minimization and the maximization versions since it generalizes the knapsack problem. We present fully polynomial approximation schemes which generate a $(1+\varepsilon)$-approximation and a $(1-\varepsilon)$-approximation for the minimization and maximization versions respectively, in $O(n^2/\varepsilon)$ time. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Facility location; Tree-shaped facility; Knapsack problems

1. Introduction

Let $T=(V,E)$ be an undirected tree with node set $V=\{v_1,\dots,v_n\}$ and edge set E. Each edge is assumed to be rectifiable. In particular, an edge is identified as a unit interval so that we can refer to its interior points. We assume that T is embedded in the Euclidean plane. Let $A(T)$ denote the continuum set of points on the edges of T. We view $A(T)$ as a connected and closed set which is a union of $(n-1)$ unit intervals. Let $P(v_i,v_j)$ denote the (unique) simple path in $A(T)$ connecting v_i and v_j. Suppose that T is rooted at v_1. For each node v_j, $j=2,\dots,n$, let $f(v_j)$, the *parent* of v_j, be the node $v\in V$, closest to v_j, $v\neq v_j$, on $P(v_1,v_j)$. With the above notation we set $E=\{e_2,\dots,e_n\}$, where e_j, $j=2,\dots,n$ is the edge connecting v_j with its parent $f(v_j)$. A point (x,j), $0\leqslant x\leqslant 1$, on e_j is represented by its Euclidean distance x from $f(v_j)$. Specifically, the endpoints (nodes) of e_j, $f(v_j)$ and v_j, are represented by $(0,j)$ and $(1,j)$, respectively.

A subset $Y\subseteq A(T)$ is a *subtree* of T if Y is both connected and closed. Y is also viewed as a finite (connected) collection of partial edges (closed subintervals), such that the intersection of any pair of distinct partial edges is empty or is a point in V.

0166-218X/98/$19.00 © 1998 Elsevier Science B.V. All rights reserved.
PII S0166-218X(98)00059-6

A subtree is said to be *discrete* if all its (relative) boundary points are nodes of T. It is *almost discrete* if at most one of its boundary points is not a node.

An edge e_j, $j=2,\dots,n$, is associated with a positive integer, a_j; a_j is the *length* of e_j. If (x,j) and (y,j) are two points on e_j the length of the partial edge connecting them is $a_j|x-y|$.

The numbers $a_2,\dots,a_n$ induce a distance function on $A(T)$. If (x,i) and (y,j) are two points consider the unique simple path in $A(T)$ connecting them. This path is viewed as a collection consisting of edges and at most two partial edges. The distance between (x,i) and (y,j) is the sum of the lengths of the edges and partial edges on the path.

Similarly, if Y is a subtree, the length of Y, $L(Y)$, is the sum of the lengths of its edges and partial edges.

Finally we define, $d(v_j,Y)$, the distance between a node $v_j\in V$ and a subtree $Y\subseteq A(T)$, to be the distance of v_j to the closest point in Y. In particular, if v_j is in Y then $d(v_j,Y)=0$.

In this paper we discuss the problem of locating a tree-shaped facility (a subtree) of a given length in a tree network, with the objective of optimizing the weighted sum of node distances from this facility. To define the problem formally, suppose that each node v_j, $j=1,\dots,n$, is associated with a nonnegative integer weight w_j. For each subtree $Y\subseteq A(T)$ define

$$F(Y)=\sum_{j=1}^{n} w_j d(v_j,Y).$$

Let A be a nonnegative integer satisfying $A\leqslant L(T)$. We consider the following two location models.

$$\begin{aligned}&\text{Min}\quad F(Y)\\ &\quad\text{s.t.}\quad L(Y)\leqslant A,\end{aligned}\tag{1.1}$$

Y is a subtree of T,

$$\begin{aligned}&\text{Max}\quad F(Y)\\ &\quad\text{s.t.}\quad L(Y)\geqslant A,\end{aligned}\tag{1.2}$$

Y is a subtree of T.

Note that (1.2) models the problem of locating an obnoxious facility. For example, consider the problem of locating a garbage dumping area of a given size A along a highway system [14].

When $A=0$ in the above problems, an optimal subtree must be a point. Thus, (1.1) and (1.2) reduce to the classical median and antimedian problems, respectively. In particular, $O(n)$ algorithms are available for solving these models [5, 15].

Several instances of problems (1.1) and (1.2) have been studied in the literature, [6, 7, 11, 12]. Minieka [11] and Hakimi et al. [6] showed that there are optimal subtrees for (1.1) and (1.2) which are almost discrete. Minieka presented a greedy-type polynomial algorithm to solve problem (1.1). Rabinovitch and Tamir [12] proved that problem (1.2) is NP-hard and gave a pseudopolynomial algorithm for its solution. Hakimi et al. [6] considered also the discrete versions of (1.1) and (1.2), where the selected subtree facility is further restricted to be discrete. They observe that the discrete versions are NP-hard since they generalize respectively, the minimization and the maximization knapsack problems.

In this paper we focus on the above discrete problems and present fully polynomial approximation algorithms. We note that such algorithms for the knapsack problem have already been presented in the literature [4, 10, 13]. Thus, our results can also be viewed as a generalization of these studies.

Let F^* denote the optimal objective value of the discrete version of (1.1) ((1.2)). Let ε be a positive number. A discrete subtree Y is called a $(1+\varepsilon)$-*approximation solution* ($(1-\varepsilon)$-*approximation solution*) if it satisfies the constraints of (1.1) ((1.2)), and $F(Y)\leqslant(1+\varepsilon)F^*$ ($F(Y)\geqslant(1-\varepsilon)F^*$).

The algorithms we present generate a $(1+\varepsilon)$-approximation and a $(1-\varepsilon)$-approximation for the minimization and maximization versions respectively, in $O(n^2/\varepsilon)$ time. The algorithms are based on an application of the interval partitioning method suggested in [13]. A main ingredient of this method is a dynamic programming algorithm which solves the given problem exactly in pseudopolynomial time. For that purpose we implement the "left–right" dynamic programming technique of [8]. Our main contribution in this general approach is the preprocessing, where we improve upon the obvious factor of n heuristics and efficiently compute a 2-approximation and a 1/2-approximation for the minimization and maximization problems respectively. These improvements yield a running time speedup of a factor of n over the straightforward "left–right" approximation scheme, from $O(n^3/\varepsilon)$ to $O(n^2/\varepsilon)$.

1.1. Notation and preprocessing

To facilitate the discussion we introduce the following notation. Suppose that the tree $T=(V,E)$ is rooted at node v_1, and let $v_1,v_2,\ldots,v_n$ be a depth-first ordering of the nodes in V. For each node v_j in V define V_j, the set of *descendants* of v_j, to be the set of nodes v in V having v_j on the unique path connecting them to the root v_1. Define S_j, the set of *children* of v_j, to be the subset of descendants of v_j that are connected to v_j with an edge. Note that v_j is in V_j but not in S_j. Set $s_j=|S_j|$. For any t, $t=0,\ldots,s_j$, let $T[j,t]$ be the subtree of T induced by v_j, the first t children (in order of index) of v_j, and all the descendants of these t children. Similarly let $T'[j,t]$ be the subtree of T induced by $T[j,t]$ and all nodes in V with indices lower than that of v_j. Note that $T'[j,t]$ is a subtree rooted at v_1. Let $V[j,t]$ and $V'[j,t]$ denote the node sets of $T[j,t]$ and $T'[j,t]$, respectively.

Let T' be a discrete subtree of T containing the root v_1. A node v_j of T' is called a *leaf* of T', if it has no children in T'. The edge e_j, connecting v_j to its parent $f(v_j)$, is then called a *leaf edge* of T'. For each node v_j in V define

$$W_j = \sum_{v_i \in V_j} w_i, \quad A_j = \sum_{v_i \in V_j} a_i \quad \text{and} \quad D_j = \sum_{v_i \in V_j} w_i d(v_i, v_j).$$

For convenience we set $a_1 = 0$. For each $j = 2, \dots, n$, we also define D_j^+ to be the sum of weighted distances of the nodes in V_j to the parent of v_j, $f(v_j)$. Thus, $D_j^+ = D_j + W_j a_j$. We obtain the following recursive equations:

$$W_j = w_j + \sum_{v_k \in S_j} W_k, \quad A_j = a_j + \sum_{v_k \in S_j} A_k \quad \text{and} \quad D_j = \sum_{v_k \in S_j} (D_k + W_k a_k).$$

Starting with the leaves of the tree we recursively compute W_j, A_j, D_j and D_j^+, for all $j = 1, \dots, n$, in $O(n)$ time.

2. The minimization model

In this section we discuss the discrete version of (1.1). We start by considering the following restricted version, where the selected subtree must contain a distinguished node.

$$\begin{aligned} \text{Min} \quad & F(Y) = \sum_{j=1}^{n} w_j d(v_j, Y) \\ \text{s.t.} \quad & L(Y) \leqslant A, \end{aligned} \tag{2.1}$$

Y is a discrete subtree of T containing v_1.

We note in passing that (2.1) generalizes the knapsack minimization problem, and therefore it is NP-hard.

Let $WD(A)$ denote the optimal objective value of (2.1). Given $\varepsilon > 0$, we present an $O(n^2/\varepsilon)$ algorithm that finds a $(1 + \varepsilon)$-approximation for problem (2.1). (Such an algorithm is called a fully polynomial approximation scheme [2].) For comparison purposes the fastest known algorithm for the knapsack minimization problem is also of the same complexity [4]. We start by producing a 2-approximation solution.

2.1. A 2-approximation

Consider first the relaxation of (2.1) where the selected subtree is not restricted to be discrete. This relaxed problem has a greedy-type polynomial algorithm which is the natural extension of the algorithm in [11] for the case where v_1 is the centroid of the tree and $w_j = 1$, $j = 1, \dots, n$. (Recall that a node $v \in V$ is a weighted centroid of T if no connected component obtained from T by the removal of v has a total node weight exceeding $\sum_{j=1}^{n} w_j/2$.)

The Greedy Algorithm

$Y(A)$ will denote an optimal solution to the relaxation of (2.1) where the selected subtree is not required to be discrete.

Step 0: Set $i = 1$, $A^1 = 0$, $Y(A^1) = \{v_1\}$.

Step 1: Let $E_i \subseteq E$ be the set of all edges that are adjacent to $Y(A^i)$, i.e., have exactly one node in $Y(A^i)$.

Step 2: Select an edge $e_{j(i)}$ in E_i such that $W_{j(i)} = \text{Max}_{\{e_j \in E_i\}}\{W_j\}$.

Step 3: Set $A^{i+1} = A^i + a_{j(i)}$, and let $Y(A^{i+1})$ be the subtree defined as the union of $Y(A^i)$ and the points $(x, j(i))$, $0 \leqslant x \leqslant 1$, on $e_{j(i)}$.

Step 4: If $i + 1 = n$ stop. Otherwise set $i \leftarrow i + 1$ and go to Step 1.

We claim that the greedy algorithm correctly finds an optimal subtree for the relaxed problems with $A = A^i$, $i = 1, \ldots, n$. (Note that $A^n = \sum_{e_j \in E} a_j$.) Furthermore, let A be a positive number and $A^i \leqslant A < A^{i+1}$ for some $i = 1, \ldots, n-1$. Then the optimal subtree for the relaxed problem is the union of $Y(A^i)$ and the set of points $(x, j(i))$, $0 \leqslant x \leqslant (A - A^i)/a_{j(i)}$, on $e_{j(i)}$.

The validity proof of the greedy algorithm can be derived from a reformulation of the problem as a continuous knapsack problem. First, each edge e_j in E is associated with a variable x_j. x_j is bounded between 0 and 1, and it represents the partial edge of e_j extending from $f(v_j)$ to the point (x_j, j) on e_j. The relaxation of problem (2.1) can now be written as

$$\begin{aligned} \text{Min} \quad & \sum_{j=2}^{n} W_j a_j (1 - x_j) \\ \text{s.t.} \quad & \sum_{j=2}^{n} a_j x_j \leqslant A, \\ & 0 \leqslant x_j \leqslant 1, \quad j = 2, \ldots, n, \end{aligned} \tag{2.2}$$

$$\text{if } f(v_j) = v_i,\ i \neq 1, \text{ and } x_j > 0, \text{ then } x_i = 1,\ j = 2, \ldots, n.$$

Deleting the last constraint, we obtain the following continuous maximum knapsack problem.

$$\begin{aligned} \text{Max} \quad & \sum_{j=2}^{n} W_j a_j x_j \\ \text{s.t.} \quad & \sum_{j=2}^{n} a_j x_j \leqslant A, \\ & 0 \leqslant x_j \leqslant 1, \quad j = 2, \ldots, n. \end{aligned} \tag{2.3}$$

From the nature of this particular continuous knapsack problem, it follows that an optimal solution can be obtained by first ranking the variables according to descending order of the $\{W_j\}$ coefficients, and then, following this ranking, successively assigning the largest possible values to the variables. In such a solution all variables but possibly one, will be equal to 1. From the definition of the $\{W_j\}$ coefficients it follows that if v_i is the parent of v_j, then $W_i \geqslant W_j$. Thus, we can assume without loss of generality that the optimal solution to (2.3), generated above, coincides with the solution generated by the Greedy Algorithm. In particular, it is also feasible, and therefore, optimal to problem (2.2). This validates the Greedy Algorithm.

The above algorithm finds the best subtree containing a distinguished node, v_1. It can easily be modified to find an optimal subtree containing any prespecified connected and closed subset of $A(T)$. This is achieved by first contracting the given subset to one of its points, and then applying the above algorithm.

The Greedy Algorithm can be implemented in $O(n \log n)$ time. Throughout the algorithm we maintain the numbers $\{W_j\}$, $e_j \in E_i$, in a heap. Since each edge enters the set E_i at some iteration i and departs at a later iteration, the total number of insertions and deletions performed on the heap is $2(n-1)$. Thus, the total effort is $O(n \log n)$.

We note that problem (2.3) can actually be solved in linear time, by using the linear time algorithm of [1] for the continuous knapsack problem. However, in the 2-Approximation Algorithm we will need the ranking of the variables (edges), produced by the Greedy Algorithm. We are now ready to present a scheme for generating a 2-approximation.

A 2-approximation algorithm

The following 2-phase scheme generates a collection S of at most $(n-1)$ feasible subtrees to problem (2.1). We will show that the best of these solutions constitutes a 2-approximation solution. To understand the formal description of the procedure, note that in the first phase we apply the Greedy Algorithm until we generate the first subtree Y^1, and the first critical edge, i.e., until we exceed for the first time the length upper bound A. The second phase generates a collection $S=\{Y^1, Y^2, \ldots, Y^t\}$, $t \leqslant n-1$, of subtrees and a respective sequence $E^c=\{e_{m(1)}, e_{m(2)}, \ldots, e_{m(t)}\}$ of edges that we call *critical*. The t subtrees in S satisfy the constraints of (2.1). We will prove that $\mathrm{Min}_{\{k=1,\ldots,t\}}\{F(Y^k)\} \leqslant 2WD(A)$, where $WD(A)$ is the optimal objective value of (2.1).

Since we will deal only with discrete subtrees, in the sequel we refer to a subtree as a set of edges that satisfy the connectivity property. We assume without loss of generality that $L(T)>A$, since otherwise the approximation algorithm will output T itself and thus be optimal.

Phase I

Step 0: Set $i=1$, $X^1=\{v_1\}$, $A^1=0$, and $E^1=\emptyset$.

Step 1: Define $E_i \subseteq E$ to be the set of all edges that have exactly one node in X^i.

Step 2: Select an edge $e_{j(i)}$ in E_i such that $W_{j(i)}=\mathrm{Max}_{\{e_j \in E_i\}}\{W_j\}$. Insert the edge $e_{j(i)}$ into E^1.

Step 3: Set $A^{i+1}=A^i+a_{j(i)}$. If $A^{i+1}>A$ proceed to Step 4. If $A^{i+1}<A$ let X^{i+1} be the union of X^i and the set of points on $e_{j(i)}$. Set $i \leftarrow i+1$ and go to Step 1. If $A^{i+1}=A$ let Y^* be the union of X^i and the set of points on $e_{j(i)}$. Stop. (Y^* is an optimal solution of problem (2.1).)

Step 4: Define $Y^1=X^i$ and $m(1)=j(i)$. Stop.

Let $E^1 = \{e_{j(1)}, \ldots, e_{j(p)}\}$, $(e_{j(p)} = e_{m(1)})$, be the sequence of edges generated during the first phase. This phase also outputs the subtree Y^1, which is feasible to problem (2.1). Y^1 is the first subtree in the collection S. Recall that Y^1 consists of the edges $\{e_{j(1)}, \ldots, e_{j(p-1)}\}$. In the second phase we scan the sequence of edges E^1 backwards, and delete certain edges to generate at most $p-1$ additional subtrees.

Phase II

Step 0: Set $E^1 = \{e_{j(1)}, \ldots, e_{j(p)}\}$, $E^c = \{e_{m(1)}\}$, $k = 2$, $i = 1$, $A' = \sum_{e_j \in E^1} a_j$, and $F = F(Y^1) - W_{j(p)} a_{j(p)}$. Go to Step 4.

Step 1: If the edge $e_{j(p-i)}$ is a leaf edge of T^1, and $A' - a_{j(p-i)} > A$, delete the edge $e_{j(p-i)}$ from E^1, set $A' \leftarrow A' - a_{j(p-i)}$, $F \leftarrow F + W_{j(p-i)} a_{j(p-i)}$ and $i \leftarrow i + 1$. Go to Step 4.

Step 2: If the edge $e_{j(p-i)}$ is a leaf edge of T^1, and $A' - a_{j(p-i)} \leqslant A$, define $m(k) = j(p-i)$ and $Y^k = E^1 - \{e_{m(k)}\}$. Insert the edge $e_{m(k)}$ into E^c. Let $F(Y^k) = F + W_{j(p-i)} a_{j(p-i)}$. Set $k \leftarrow k + 1$, and $i \leftarrow i + 1$. Go to Step 4.

Step 3: If the edge $e_{j(p-i)}$ is not a leaf edge of T^1, set $i \leftarrow i + 1$, and go to Step 4.

Step 4: Let T^1 be the subtree whose edge set is E^1. If $p - i = 0$ stop. Otherwise, go to Step 1.

The second phase generates a collection $S = \{Y^1, \ldots, Y^t\}$, $t \leqslant p$, of feasible subtrees and a respective sequence $E^c = \{e_{m(1)}, \ldots, e_{m(t)}\}$ of edges that we call *critical.* The subtrees in S satisfy the following nestedness property. For each $k = 1, \ldots, t-1$, $Y^{k+1} \subseteq Y^k \cup \{e_{m(k)}\}$. The algorithm also outputs the sequence of values $\{F(Y^1), \ldots, F(Y^t)\}$.

Proposition 2.1. *Let Y^* be an optimal subtree solving problem* (2.1). *Let E^c be the set of critical edges produced by the* 2-*Approximation Algorithm. Then there exists a critical edge in E^c which is not contained in Y^*.*

Proof. The subtree T^1 defined at the end of Phase II is the minimal subtree containing the root v_1 and the entire set of critical edges. Since the algorithm stops at this iteration we must have $L(T^1) > A$. Thus, if Y^* is assumed to contain all critical edges we have $Y^* \supseteq T^1$, $L(Y^*) \geqslant L(T^1) > A$, which in turn contradicts the feasibility of Y^*. □

Proposition 2.2. *For $k = 1, \ldots, t$, let $E^c_k = \{e_{m(1)}, \ldots, e_{m(k)}\}$. Set $E^c_0 = \emptyset$. Define T^k_+ as the subtree consisting of Y^k and the set of points on the critical edge $e_{m(k)}$. Let Y be any subtree containing v_1 and the critical edges in E^c_{k-1}. If $L(Y) \leqslant L(T^k_+)$ then $F(Y) \geqslant F(T^k_+)$.*

Proof. At each iteration k of Phase II, we delete an edge having the lowest possible value of W, which is not in the minimal subtree containing the root v_1, and the edges in E^c_{k-1}. Therefore, it follows from the validity of the Greedy Algorithm and the definition of T_+^k, that T_+^k has the lowest possible value of the objective F amongst all subtrees Y (not necessarily discrete) containing v_1 and E_{k-1}^c, and satisfying $L(Y) \leqslant L(T_+^k)$. □

Theorem 2.3. *Let* $\{Y^1, \ldots, Y^t\}$, $t \leqslant p$, *be the collection of subtrees generated by the algorithm. Then* $\operatorname{Min}_{\{k=1,\ldots,t\}}\{F(Y^k)\} \leqslant 2WD(A)$.

Proof. Let Y^* be an optimal subtree solving problem (2.1), i.e. $F(Y^*) = WD(A)$. Using Proposition 2.1, we let k be such that Y^* contains the critical edges $e_{m(1)}, \ldots, e_{m(k-1)}$, but not $e_{m(k)}$. Let T_+^k be defined as in Proposition 2.2. Since we have $L(T_+^k) > A \geqslant L(Y^*)$ it follows from Proposition 2.2 that

$$F(T_+^k) \leqslant F(Y^*). \tag{2.4}$$

From the definitions,

$$F(T_+^k) = F(Y^k) - W_{m(k)} a_{m(k)}. \tag{2.5}$$

Since $e_{m(k)}$ is not contained in Y^*, we have

$$F(Y^*) \geqslant W_{m(k)} a_{m(k)}. \tag{2.6}$$

Thus, combining (2.4)–(2.6) we conclude that

$$2F(Y^*) \geqslant F(T_+^k) + W_{m(k)} a_{m(k)} = F(Y^k). \qquad \square$$

It is easy to observe that the 2-Approximation Algorithm can be implemented in $O(n \log n)$ time, since the complexity bounds of Phases I and II are $O(n \log n)$ and $O(n)$, respectively.

2.2. $(1 + \varepsilon)$ *approximation schemes*

In this section we present a fully polynomial approximation scheme for problem (2.1). Specifically, we present an algorithm which given an instance of the problem and a positive ε, generates in $O(n^2/\varepsilon)$ time, a subtree Y such that $L(Y) \leqslant A$ and $F(Y) \leqslant (1+\varepsilon) WD(A)$. We apply the interval partitioning approach suggested in [13]. To implement this approach we first introduce a dynamic programming algorithm which solves problem (2.1) in pseudopolynomial time. This algorithm is based on the non-standard "left–right" approach of [8]. It solves (2.1) in $O(n \operatorname{Min}\{A, WD(A)\})$ time. (Alternatively we could have used two other dynamic programming algorithms. The first is based on the standard "bottom-up" approach (see [12]), and it solves (2.1) in $O(n(\operatorname{Min}\{A, WD(A)\})^2)$. The second algorithm is based on the nonstandard "bottom-up" approach of [3]. Its running time is $O(n \operatorname{Min}\{A, WD(A)\})$.)

The left–right approach of [8] uses the subtrees $T'[j,t]$, defined in Section 1.1. Following [8] we order these subtrees such that $T'[j,t]$ precedes $T'[j,t+1]$ for all nodes v_j and $t = 0, 1, \ldots, s_j$, and so that if $v_{j(t)}$ is the tth child of v_j, then $T'[j,t-1]$

precedes $T'[j(t),0]$ and $T'[j(t),s_{j(t)}]$ precedes $T'[j,t]$. For each triple $[j,t,L]$ define problem (2.7):

$$\begin{aligned} \text{Min} \quad & \sum_{v_i \in V'[j,t]} w_i d(v_i, Y) \\ \text{s.t.} \quad & L(Y) \leqslant L, \end{aligned} \tag{2.7}$$

Y is a discrete subtree of $T'[j,t]$ containing v_1 and v_j.

We note that with the above definitions, although $T'[j,t]$ and $T'[j(t),s_{j(t)}]$ are identical as trees, a solution to the problem corresponding to the second must include $v_{j(t)}$, while a solution to the problem corresponding to the first may not. Let $g[j,t,L]$ be the optimal solution value to problem (2.7). For each pair $[j,t]$ we maintain a (sorted) list $G[j,t]$ of pairs $(g[j,t,L],L)$, $L \leqslant A$ and $g[j,t,L] \leqslant D$, where D is some known precomputed upper bound on the objective value of (2.1). (Note that g is a nonincreasing function of L, and therefore the ordering of the pairs is well defined.) The list will consist of the nondominated pairs only. Thus, its size will be O(Min $\{A,D\}$).

The following recursive algorithm, which we call the *Left–Right Algorithm* generates all these lists.

1. If $j=1$ and $t=0$, then $G[j,t]=\{(0,0)\}$.
2. If $j>1$ and $t=0$ suppose that v_j is the kth child of v_i. Consider the list $G[i,k-1]$. The list $G[j,0]$ is obtained from $G[i,k-1]$ by first adding the constant a_j to the L component of each pair in $G[i,k-1]$, and then omitting from this list all pairs for which the L component is larger than A.
3. If $t>0$ and $v_{j(t)}$ is the tth child of v_j then the list $G[j,t]$ is generated as follows. Let $G'[j,t-1]$ be the list obtained from $G[j,t-1]$ by adding the constant $D^+_{j(t)}$ to the g coordinate of each pair in $G[j,t-1]$. Delete from $G'[j,t-1]$ those pairs with a g coordinate exceeding D. Next, let G be the list of pairs obtained by merging the list $G'[j,t-1]$ with the list $G[j(t),s_{j(t)}]$, according to the value of the L component. Finally, delete all dominated pairs, i.e., if two pairs (g^1,L^1) and (g^2,L^2) in G satisfy $L^1 \leqslant L^2$ and $g^1 \leqslant g^2$, delete the pair (g^2,L^2). $G[j,t]$ is defined as the resulted list.

The optimal value of problem (2.1) is given by the smallest g component of a pair in the list $G[n,s_n]$. (Note that the node v_n is a leaf of T, and therefore $s_n=0$.) It is easily observed that the time needed to compute any list $G[j,t]$ is O(Min $\{A,D\}$). Therefore, the total time to solve problem (2.1) by the above algorithm is O(n Min $\{A,D\}$). From the results in Section 2.1 we can compute in O($n \log n$) time a value of D which is at most twice the optimal value of problem (2.1). Thus, problem (2.1) can be solved in O(n Min $\{A, WD(A)\}$) time, where $WD(A)$ is the optimal value of (2.1). We note in passing that the O(nA) and O(nD) bounds can also be achieved by using the nonstandard bottom-up dynamic programming approach in [3], designed to solve the knapsack problem with in-tree precedence constraints.

The above dynamic programming methods which solve problem (2.1) exactly can be used by the interval partitioning method in [13] to yield a fully polynomial approximation scheme. The time bound for generating a $(1+\varepsilon)$ approximation with these approaches is $O(n^2/\varepsilon)$. For the sake of brevity we present the details only for the approach in [8].

The $(1+\varepsilon)$-Approximation Algorithm

Let $F^0(A)$ be the 2-approximation value for (2.1) computed in Section 2.1. Given a positive ε, we partition the interval $[0, F^0(A)]$ into $\lceil 2n/\varepsilon \rceil$ consecutive subintervals, *cells*, each but possibly the last one of length $\varepsilon F^0(A)/2n$. The approximation algorithm follows the steps of the above Left–Right Algorithm. For each pair $[j,t]$ the algorithm produces a list $H[j,t]$ of at most $\lceil 2n/\varepsilon \rceil$ subtrees of $T'[j,t]$, $\{Y^i\}$, containing v_1 and v_j such that the objective value of Y^i with respect to problem (2.7), $F(Y^i)$, is in the ith cell. In general, any subtree Y will be recorded by the respective pair $(F(Y), L(Y))$. In the first step of the algorithm where $j=1$ and $t=0$, the list $H[1,0]$ contains only the pair $(0,0)$ corresponding to the subtree consisting of the node v_1 only. Recursively, suppose that $t=0$ and v_j, $j>1$, is the kth child of v_i. Consider the list $H[i,k-1]$. The list $H[j,0]$ is obtained from $H[i,k-1]$ by adding the constant a_j to the L component of each pair and then removing all pairs such that the new value of L is greater than A. Next suppose that $t>0$ and let $v_{j(t)}$ be the tth child of v_j. The list $H[j,t]$ is generated as follows. Let $H'[j,t-1]$ be the list obtained from $H[j,t-1]$ by adding the constant $D^+_{j(t)}$ to the F coordinate of each pair (F,L) in $H[j,t-1]$. Delete from $H'[j,t-1]$ those pairs with an F coordinate exceeding $F^0(A)$. Place each of the remaining pairs into the appropriate cell of the interval $[0, F^0(A)]$. Let H be the union of $H'[j,t-1]$ and $H[j(t), s_{j(t)}]$. Each cell of the interval $[0, F^0(A)]$ contains at most two F values corresponding to two pairs in H. If a cell contains exactly two then remove from the list H that pair with the higher L coordinate. Thus, H will contain at most $\lceil 2n/\varepsilon \rceil$ pairs, one for each cell of the interval $[0, F^0(A)]$. Define $H[j,t]$ to be equal to H. The algorithm terminates with the final list $H[n,0]$ corresponding to the leaf node v_n. Consider a pair (F^*, L^*) in this list with the smallest F coordinate. Let Y' be the respective subtree. The claim is that Y' is a $(1+\varepsilon)$-approximation solution, i.e., $F^* \leqslant (1+\varepsilon)WD(A)$.

To validate the claim we first define an optimal solution to a subproblem (2.7) defined by the triple $[j,t,L]$ to be relevant if $L \leqslant A$, L is not smaller than the length of the path connecting v_1 and v_j, and the objective value is at most $F^0(A)$. It is clear that only relevant solutions should be considered for optimizing (2.1). Indeed, in the above approximation algorithm only the relevant solutions of the $O(n)$ subproblems $[j,t]$ are represented in the lists. If subproblem $[j,t]$ is processed at the kth step of the algorithm and $(F(Y), L(Y))$ is any one of its relevant solutions, then it is represented by some pair (F,L) in the list $H[j,t]$ where $|F(Y)-F| \leqslant k\varepsilon F^0(A)/2n$ and $L(Y) \leqslant L$. (At every step of the algorithm we introduce an additive error term of $\varepsilon F^0(A)/2n$ whenever we delete a pair corresponding to a cell containing exactly two elements.)

This proves that the pair (F,L) that we select in the last list $H[n,0]$ satisfies $L \leqslant A$ and $F \leqslant WD(A) + \varepsilon n F^0(A)/2n \leqslant (1+\varepsilon)WD(A)$, since $F^0(A) \leqslant 2WD(A)$. Thus, Y' is a $(1+\varepsilon)$-approximation solution.

We have presented an $O(n^2/\varepsilon)$ algorithm to obtain a $(1+\varepsilon)$-approximation solution to problem (2.1). However, (2.1) is a restriction of the original problem, since the optimal subtree was restricted to include a distinguished node, namely v_1. We can clearly solve the original problem by solving n restricted problems. In the jth restricted problem, the subtree is required to contain node v_j, $j = 1, \ldots, n$. This approach takes $O(n^3/\varepsilon)$ effort. There is, however, a better implementation, which is based on a divide and conquer approach, [12].

Suppose without loss of generality that v_1 is a centroid of the original tree T, i.e., no connected component of T, obtained by the removal of v_1, contains more than $n/2$ nodes. If the optimal subtree does not include v_1, it is contained in a component having at most $n/2$ nodes. Hence, it is sufficient to approximate the problem where the optimal subtree must include v_1, and then make recursive calls to problems of size at most $n/2+1$. This fact implies that the total effort of obtaining a $(1+\varepsilon)$-approximation for the unrestricted discrete version of problem (1.1) is also $O(n^2/\varepsilon)$.

3. The maximization model

In this section we briefly discuss the solution of the maximization version of (2.1).

$$\begin{aligned} \text{Max} \quad & F(Y) = \sum_{j=1}^{n} w_j d(v_j, Y) \\ \text{s.t.} \quad & L(Y) \geqslant A, \end{aligned} \tag{3.1}$$

Y is a discrete subtree of T containing v_1.

Let $WD(A)$ denote the optimal objective value of (3.1). We note that the left–right dynamic programming algorithm of [8], and the nonstandard bottom-up algorithm of [3], mentioned in Section 2, can easily be modified to solve problem (3.1). The complexity bound for (3.1) obtained with these approaches is $O(n \operatorname{Min}\{A, D\})$, where D is some precomputed known upper bound on $WD(A)$. We will show next how to find in polynomial time a feasible solution Y to (3.1) such that $WD(A) \leqslant 2F(Y)$. Setting $D = 2F(Y)$, we note that D is bounded between $WD(A)$ and $2WD(A)$. After we derive such a solution we can mimic the approach in Section 2 and modify it to find a $(1-\varepsilon)$-approximation solution to (3.1) in $O(n^2/\varepsilon)$ time for every ε less than 1. Recall that a feasible solution Y to (3.1) is a $(1-\varepsilon)$-approximation if $F(Y) \geqslant (1-\varepsilon)WD(A)$.

3.1. A 1/2-approximation

First we note that we cannot apply the approach in Section 2.1. That approach is based on an efficient algorithm to solve the relaxation of the minimization problem (2.1), where the selected subtree is not restricted to be discrete. The respective relaxation of the maximization model (3.1) is NP-hard [12].

Given a subtree Y containing v_1 and $L(Y) \geqslant A$, let $N(Y) = \{v_{j(1)}, \ldots, v_{j(p)}\}$ denote the set of nodes of $V \backslash Y$ that are connected to Y with an edge. These nodes are called the *neighbors* of Y. Then we have,

$$\sum_{k=1}^{p} A_{j(k)} \leqslant \bar{A} \quad \text{and} \quad F(Y) = \sum_{k=1}^{p} D^{+}_{j(k)},$$

where $\bar{A} = A_1 - A$. (Recall from Section 1.1 that $A_1 = \sum_{j=2}^{n} a_j$.)

Thus, Y is fully characterized by its neighbors and we can use the above expressions to reformulate (3.1).

$$\begin{aligned} \text{Max} \quad & \sum_{v_k \in S} D_k^{+} \\ \text{s.t.} \quad & \sum_{v_k \in S} A_k \leqslant \bar{A}, \\ & \text{where } S \subseteq V \text{ is a subset of distinct nodes} \\ & \text{such that no node in } S \text{ is a descendant} \\ & \text{of another node in } S. \end{aligned} \tag{3.2}$$

We assume without loss of generality that $\bar{A}$ is smaller than A_1 since otherwise A in (3.1) is nonpositive and the optimal solution is trivial. If A_j is greater than $\bar{A}$ then node v_j cannot be in any feasible solution to (3.2). Thus, we assume that A_j is ∞ whenever A_j is greater than $\bar{A}$.

The following heuristic for problem (3.2) is based on a simple intuitive greedy approach. At each step we select a node with the highest contribution to the objective per unit of the constraint (resource).

A 1/2-Approximation Algorithm for Problem (3.1)

Step 0: Set $A' = 0$, $D' = 0$ and let $V' = \emptyset$ and $V'' = V - \{v_1\}$. For $j = 2, \ldots, n$, let $C_j = D_j^{+}/A_j$, $A'_j = A_j$ and $D'_j = D_j^{+}$.

Step 1: Select a node v_j in V'' with a largest C_j coefficient. Delete v_j and all its descendants from V''.

Step 2: If $A' + A'_j \leqslant \bar{A}$, insert v_j into V' and delete all the descendants of v_j from V'. Add A'_j to A', add D'_j to D' and go to Step 3. Otherwise, stop.

Step 3: For each node $v_k \in V''$ having v_j as one of its descendants subtract A'_j from A'_k, subtract D'_j from D'_k and redefine C_k by $C_k = D'_k/A'_k$. Go to Step 1.

Let $V' = \{v_{j(1)}, \ldots, v_{j(p)}\}$ be the subset of nodes which is output by the algorithm and let v_j be the respective node which has resulted in the termination of the algorithm in Step 2. Note that V' is feasible for problem (3.2). Without loss of generality we may assume that $A_j \leqslant \bar{A}$ since otherwise $A_j = \infty$, and by the choice of v_j in Step 1, $A_k = \infty$ for every node v_k which is not a descendant of a node in V' so that the solution defined by V' is optimal. Define $C = \text{Maximum}\{D_i^{+} \mid i = 2, \ldots, n, \text{ and } A_i \leqslant \bar{A}\}$.

The claim is that $D = \text{Maximum}\{\sum_{j=1}^{p} D^+_{j(k)}, C\}$ is the value of a 1/2-approximation solution for (3.2). First note that D does indeed correspond to a feasible solution to (3.2). To verify that $WD(A) \leqslant 2D$ it will suffice to prove the statement that if $V'(j)$ is the set obtained from V' by deleting all the descendants of v_j, then $V'(j) \cup \{v_j\}$ optimally solves problem (3.2) with $A'' = \sum_{v_{j(k)} \in V'(j)} A_{j(k)} + A_j$ replacing $\bar{A}$. (Note that $A'' > \bar{A}$.)

Indeed if the latter statement holds then

$$\begin{aligned} WD(A) \leqslant WD(A'') &= \sum_{v_{j(k)} \in V'(j)} D^+_{j(k)} + D^+_j \leqslant \sum_{k=1}^{p} D^+_{j(k)} + D^+_j \\ &\leqslant D + C \leqslant 2D. \end{aligned}$$

Formally, we need to prove the following proposition. (We will show that the heuristic provides an optimal solution to a relaxation of (3.2) with A'' replacing $\bar{A}$.)

Proposition 3.1. *Let $V'_t = \{v_{j(1)}, \ldots, v_{j(t)}\}$ be the subset of nodes generated by the algorithm at the end of the t-th iteration. Then V'_t maximizes* (3.2) *with $\bar{A} = \sum_{k=1}^{t} A_{j(k)}$.*

Proof. We use the following integer programming formulation for (3.2):

$$\begin{aligned} \text{Max} \quad & \sum_{j=2}^{n} W_j a_j x_j \\ \text{s.t.} \quad & \sum_{j=2}^{n} a_j x_j \leqslant \bar{A}, \\ & x_i \leqslant x_j \text{ if } v_j \text{ is a descendant of } v_i \text{ (for every pair of nodes } v_i \text{ and } v_j), \\ & x_j \in \{0,1\}, \quad j = 2, \ldots, n. \end{aligned} \tag{3.3}$$

(Note that in the above formulation, $x_j = 1$ if and only if the node v_j is a descendant of some node in the selected set S.)

Consider the linear programming relaxation of (3.3) obtained by replacing the integer binary constraints by restricting the variables to be between 0 and 1. Let j be an index satisfying $D^+_j / A_j \geqslant D^+_k / A_k$ for $k = 2, \ldots, n$. It is easy to show that for any positive value of $\bar{A}$ there is an optimal solution where $x_i = \text{Min}\{1, \bar{A}/A_j\}$ for every i such that v_i is a descendant of v_j. Arguing inductively, we then conclude that the greedy approach used in the above approximation algorithm solves the linear programming relaxation. In particular if we set $\bar{A} = \sum_{k=1}^{t} A_{j(k)}$, we get an integer solution which in turn must solve (3.2) for this particular value of $\bar{A}$. This completes the proof of the proposition. □

To summarize, we have presented a greedy algorithm which finds a 1/2-approximation solution to problem (3.1). The running time of the algorithm is certainly $O(n^2)$. As mentioned above, having found a 1/2-approximation solution we can then mimic the approach in Section 2 and find, for every ε bounded above by 1, a $(1-\varepsilon)$-approximation solution to (3.1) in $O(n^2/\varepsilon)$ time. For the sake of brevity we skip the details.

4. Final comments

We have presented above fully polynomial approximation schemes for problems (2.1) and (3.1). In these problems the selected subtree Y is restricted to be discrete. As mentioned in the Introduction, when we remove the discreteness assumption problem (2.1) can be solved in polynomial time [11], while problem (3.1) remains NP-hard [12]. In both cases there is an optimal subtree which is almost discrete. Consider the variant of (3.1) when Y is now an almost discrete subtree of T containing the distinguished node v_1. Using the standard bottom-up approach, it was shown in [12] that this problem can be solved in $O(nA^2)$ time. We note in passing that the latter bound can be improved to $O(nA)$ if we adopt the left–right dynamic programming approach. Moreover, the fully polynomial approximation scheme for the discrete case can be easily modified to find $(1-\varepsilon)$-approximations for the almost discrete case in $O(n^2/\varepsilon)$ time.

Finally, we note that the above approach can easily be modified to yield fully polynomial approximation schemes even for some nonlinear objective functions commonly used in location theory. For example, consider the following covering problem, [7]. Suppose that each node $v_j \in V$ is associated with two nonnegative integer parameters; a radius r_j and a penalty term p_j.

Define

$$f_j(Y) = \begin{cases} 0 & \text{if } d(v_j, Y) \leqslant r_j, \\ p_j & \text{otherwise.} \end{cases}$$

Thus, there is a penalty of p_j if the node v_j is not within a distance of r_j from the selected subtree Y. The objective function is to minimize the total penalty cost, $F(Y) = \sum_{j=1}^{n} f_j(Y)$, subject to a length constraint as in (1.1).

More generally, our approach can be used to obtain fully polynomial approximation schemes for any objective function $F(Y) = \sum_{j=1}^{n} f_j(Y)$, where $f_j(Y)$, $j = 1, \ldots, n$, is a nondecreasing integer-valued function (not necessarily linear or stepwise linear) of the distance $d(v_j, Y)$. It is easy to verify that the left–right procedure of Section 2.2 can be adapted to solve the general case, provided that we have an initial approximation like the one we find in Section 2.1 for the linear case. However, it is not yet clear whether the results of Section 2.1, where we obtain a 2-approximation for the linear case can be extended to the general case of nondecreasing functions of the distances. Instead, we can initiate the process for the general case by finding an n-approximation solution. Such a solution can be obtained by minimizing the objective $G(Y) = \mathrm{Max}_{\{j=1,\ldots,n\}}\{f_j(Y)\}$, since a minimizer of $G(Y)$ is an n-approximation solution for the minimum of $F(Y)$. Starting with an n-approximation solution the running time that we achieve for this general and unifying model is $O(n^3/\varepsilon)$. (The reader is referred to Labbé et al. [9] where the authors presented approximation schemes for other nonlinear versions of the knapsack problem.)

References

[1] E. Balas, E. Zemel, An algorithm for large zero-one knapsack problems, Oper. Res. 28 (1980) 1130–1154.

[2] R. Garey, D.S. Johnson, 'Strong' NP-completeness results: Motivation, examples, and implications, JACM 25 (1978) 499–508.

[3] G.V. Gens, Resource allocation in hierarchic systems, Eng. Cybernet. 22 (1984) 122–128.

[4] G.V. Gens, E.V. Levner, Computational complexity of approximation algorithms for combinatorial problems, in Lecture Notes in Computer Science, vol. 74, Springer, Berlin, 1979, pp. 292–300.

[5] A. Goldman, Optimal center location in simple networks, Transport. Sci. 5 (1971) 212–221.

[6] S.L Hakimi, E. Schmeichel, M. Labbé, On locating path – or tree-shaped facilities on networks, Networks 23 (1993) 543–556.

[7] V. Hutson, C. ReVelle, Maximal direct covering tree problems, Transport. Sci. 23 (1989) 288–299.

[8] D.S. Johnson, K.A. Niemi, On knapsacks, partitions and a new dynamic programming technique for trees, Math. Oper. Res. 8 (1983) 1–14.

[9] M. Labbé, E.F. Schmeichel, S.L. Hakimi, Approximation algorithms for the capacitated plant location problem, Oper. Res. Lett. 15 (1994) 115–126.

[10] E. Lawler, Fast approximation algorithms for knapsack problems, Math. Oper. Res. 4 (1979) 339–356.

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

[12] R. Rabinovitch, A. Tamir, On a tree-shaped facility location problem of Minieka, Networks 22 (1992) 515–522.

[13] S. Sahni, General techniques for combinatorial approximations, Oper. Res. 25 (1977) 920–936.

[14] A. Tamir, Obnoxious facility location on graphs, SIAM J. Discrete Math. 4 (1991) 550–567.

[15] S.S. Ting, A linear time algorithm for maximum facility location on tree networks, Transport. Sci. 18 (1984) 76–84.