The 2-radius and 2-radiian Problems on Trees

Hung-Lung Wang[1] and Kun-Mao Chao[1,2,3,*]

[1]Department of Computer Science and Information Engineering

[2]Graduate Institute of Biomedical Electronics and Bioinformatics

[3]Graduate Institute of Networking and Multimedia

National Taiwan University, Taipei, Taiwan 106

Abstract

In this paper, we consider two facility location problems on tree networks. One is the 2-radius problem, whose goal is to partition the vertex set of the given network into two non-empty subsets such that the sum of the radii of these two induced subgraphs is minimum. The other is the 2-radiian problem, whose goal is to partition the network into two non-empty subsets such that the sum of the centdian values of these two induced subgraphs is minimum. We propose an $O(n)$-time algorithm for the 2-radius problem on trees and an $O(n \log n)$-time algorithm for the 2-radiian problem on trees, where n is the number of vertices in the given tree.

Keywords: facility location problem, center, median, centdian, radius, radiian, tree

1 Introduction

In a facility location problem, one is asked to deploy some facilities in a given network to optimize some objectives. Depending on the requirements, facility location problems can be categorized by $\Gamma/\Delta/p$ together with an objective function and the network type, where the supply set Γ stands for the locations to deploy facilities, the demand set Δ stands for the locations of all customers, and p is the number of facilities we need to deploy. Usually, Γ and Δ are in $\{V(G), A(G)\}$, where G is the

*kmchao@csie.ntu.edu.tw

given network, $V(G)$ is the set of vertices in G, and $A(G)$ denotes all continuous positions (called points) on the edges of G. Two classic facility location problems are the center problem and the median problem. The center problem concerns the longest distance from each customer to its closest facility, and the median problem focuses on the sum of distances from all customers to their closest facilities. Both problems in general graphs for arbitrary p are NP-hard [15, 16], but are polynomial-time solvable for some special graphs, like trees and cactus networks [2, 6, 14, 17, 18, 19, 23].

In reality, people might wish to pursue more than one objectives. Thus, striking a balance between the center problem and the median problem is an intuitive extension. Halpern [8, 9] proposed a way to resolve the dilemma, namely to optimize a convex combination of the objective function of the center problem and that of the median problem. The position which minimizes the objective function is called the *centdian* of the given network, and this is called the *centdian problem*. The p-centdian problem in general graphs for arbitrary p is NP-hard since both the p-center problem and the p-median problem in general graph are NP-hard [15, 16]. On trees, the p-centdian problem is polynomially solvable for arbitrary p [23]. For $p = 1$, it can be solved in linear time [9, 23]. For $p = 2$, it can be solved in $O(n^2)$ time, where n is the number of vertices in the given tree network [21, 23]. Readers can refer to [5, 11, 20, 24] for related researches about centdian.

Proietti and Widmayer [22] mentioned that there are two different viewpoints to the facility location problems either from the customer's or facility's aspect, where the former is *customer-centric*, and the latter is *facility-centric*. In the traditional center problem, the objective is customer-centric since each customer asks for the service from the closest server. Proietti and Widmayer proposed a facility-centric problem, namely the $V(G)/V(G)/p$-radius problem, whose objective is to minimize the sum of the set-up costs of all facilities, where the set-up cost depends on the longest distance from each facility to the customers it serves. Therefore, the deployment of facilities depends on the partition of the network. In general graphs, they proposed an $O(n^{2p}/p!)$-time algorithm to solve this problem for $p > 2$, and an $O(mn^2 + n^3 \log n)$-time algorithm for $p = 2$, where m and n denote the numbers of edges and vertices, respectively, in the given graph. For trees and graphs with bounded tree width h, Bilò et. al [4] proposed an $O(n^3p^3)$-time and an $O(n^{4h+4}p^3)$-time algorithms, respectively.

The traditional median problem is customer-centric. If we view the median problem from the facility's viewpoint, the resulting partition can be obtained directly from the result of the traditional median problem as follows. The network is divided into p parts, where p is the number of facilities to deploy. Within each part there is exactly one facility, which is the closest one to the vertices in that part among all facilities. The p-centdian problem is customer-centric since the objective function consists of those of the center problem and the median problem. Inspired by [8] and [22], we propose a facility-centric problem, called the *radiian problem*, whose objective function is a convex combination of that of the radius problem and that of the facility-centric median problem.

In this paper, we assume that $\Gamma = A(G)$, and $\Delta = V(G)$. Thus when mentioning the problems, we omit these two parameters. We consider two facility location problems on tree networks, and both of them are facility-centric. One is the 2-radius problem, and the other is the 2-radiian problem. The rest of this paper is organized as follows. In Section 2, we give some preliminaries and formally define the problems. In Sections 3 and 4, an $O(n)$-time algorithm and an $O(n \log n)$-time algorithm are proposed for the 2-radius and 2-radiian problems on trees, respectively, where n is the number of vertices in the given tree. Some concluding remarks are given in Section 5.

2 Problem definitions and preliminaries

In this section, we shall define some notations used in this paper. The reader can refer to Harary [12] for any graph-theory terms not defined here.

Given is an undirected graph $G = (V(G), E(G), l, w)$ with the vertex set $V(G)$, the edge set $E(G)$, the edge length function $l : E(G) \mapsto \{x : x \in \mathbf{R} \text{ and } x > 0\}$, and the vertex weight function $w : V(G) \mapsto \{x : x \in \mathbf{R} \text{ and } x \geq 0\}$. We also use $w(G)$ to denote the sum of weights of all vertices in G, i.e. $w(G) = \sum_{v \in V(G)} w(v)$. For a *point* u in an edge, we can characterize u by a triple (e, v_1, r), which means that u is in the edge $e = (v_1, v_2)$ and the distance between u and vertex v_1 is r. Note that point u can also be characterized by $(e, v_2, l(e) - r)$. Thus the distance $d_G(u, v)$ between two points u and v in G is defined to be the length of a shortest path from u to v in G. If there is no path between u and v in G, then $d_G(u, v) = \infty$.

We denote the set of points in G by $A(G)$, and for each point $u \in A(G)$, we associate u

with the following functions: (i) the center function, which is the eccentricity of u in G and is defined to be $f_c^G(u) = \max_{x \in V(G)} d_G(x, u)$, (ii) the median function, which is defined as $f_m^G(u) = \sum_{v \in V(G)} w(v) \cdot d_G(u, v)$, and (iii) the centdian function $f_\lambda^G(u) = \left(\lambda f_m^G(u) + (1-\lambda) f_c^G(u)\right)$, where $\lambda \in [0, 1]$. The *center* of G is the point x which minimizes f_c^G. The *median* and the *centdian* can be defined in a similar way, i.e. the points which minimize f_m^G and f_λ^G, respectively. The *diameter* of G is defined to be a path whose length is equal to the maximum eccentricity among all points in the given network. Note that in a tree network with positive edge lengths, the diameter always exists and is a path with two leaves as the end points since otherwise we can stretch the path to be a longer one. We denote the subgraph of G induced by $U \subseteq V(G)$ by $G[U]$. The 2-radius and the 2-radiian problems on trees are formally defined as follows.

Definition 1: (*The 2-radius problem*) Given an undirected tree $T = (V(T), E(T), l, w)$, the 2-radius problem asks for a partition (U_1, U_2) of $V(T)$ and the centers of $T[U_1]$ and $T[U_2]$, where $U_i \neq \emptyset$ for $i \in \{1, 2\}$, $U_1 \cap U_2 = \emptyset$, and $U_1 \cup U_2 = V(T)$, such that

$$f_r^T(U_1, U_2) = \sum_{i \in \{1,2\}} \min_{u \in A(T[U_i])} f_c^{T[U_i]}(u)$$

is minimum.

Definition 2: (*The 2-radiian problem*) Given an undirected tree $T = (V(T), E(T), l, w)$, and a real $\lambda \in [0, 1]$, the 2-radiian problem asks for a partition (U_1, U_2) of $V(T)$ and the centdians of $T[U_1]$ and $T[U_2]$, where $U_i \neq \emptyset$ for $i \in \{1, 2\}$, $U_1 \cap U_2 = \emptyset$, and $U_1 \cup U_2 = V(T)$, such that

$$f^T(U_1, U_2) = \sum_{i \in \{1,2\}} \min_{u \in A(T[U_i])} f_\lambda^{T[U_i]}(u)$$

is minimum.

For a feasible partition (U_1, U_2) (i.e. $U_i \neq \emptyset$ for $i \in \{1, 2\}$, $U_1 \cap U_2 = \emptyset$, and $U_1 \cup U_2 = V(T)$), we know that if $T[U_i]$ is not connected for $i \in \{1, 2\}$, both f_r^T and f^T will be unbounded since there is an $x \in U_i$ not adjacent to the vertices in $U_i - \{x\}$ and $f_c^{T[U_i]}(x) = d_{T[U_i]}(x, k) = \infty$ for $k \in U_i$. Similarly, $f_\lambda^{T[U_i]} = \infty$ if $T[U_i]$ is not connected. Thus $T[U_i]$ must be connected for $i \in \{1, 2\}$ since $f_r^T(V(T) - \{x\}, \{x\})$ and $f^T(V(T) - \{x\}, \{x\})$ are bounded, where x is a leaf of T. However, T is a

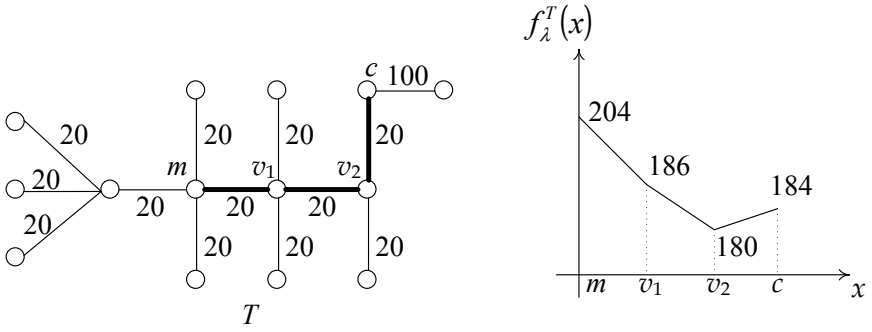

Figure 1: A tree T with unit vertex weights, and the centdian values of points on $\mathcal{P}[m, c]$ with $\lambda = 0.1$.

tree, and we know that the removal of an arbitrary edge in a tree results in exactly two connected components. Thus, $T[U_1] \cup T[U_2] = T - \{e\}$ for some $e \in E$.

Based on the above observation, the 2-radius and the 2-radiian problems on trees can be solved by first computing the pairs of centers and centdians associated with the removal of each edge of T, respectively, and then finding the optimal solutions. This method takes $O(n^2)$ time since we need $O(n)$ time to find a center as well as a centdian of a tree [8, 26]. Our algorithms are based on this method. Before introducing our algorithms, some important properties, which will be used later, are summarized as follows.

In [8], Halpern proved that the centdian of a tree T must reside in the path between the center c and the median m of T. In the following, we call this path the *candidate path* of T and denote the unique simple path in T between u and v by $\mathcal{P}[u, v]$. The length of a path $\mathcal{P}$ is denoted by $|\mathcal{P}|$. Useful properties of a centdian function are summarized in Lemma 1.

Lemma 1: [8] Given a tree T, a centdian function is a convex, continuous, and piecewise linear function of $x \in A(\mathcal{P}[m, c])$, with *breaking points* (the points where the derivative of f_λ^T changes) at the vertices on $\mathcal{P}[m, c]$.

By Lemma 1, we know that the minimum of the centdian function occurs on at least one vertex or the end points of $\mathcal{P}[m, c]$. An example is represented in Figure 1. The following properties hold for the center c of T.

Property 2: [25] Let $u, v \in V(T)$. If $d_T(v, u) = f_c^T(v)$, then $c \in A(\mathcal{P}[v, u])$.

Property 3: [26] Let $\mathcal{D} = \mathcal{P}[x, y]$ be a diameter of a tree T, and c be the center of T. We have $c \in \mathcal{D}$ and $d_T(c, x) = |\mathcal{D}|/2$.

In the following, the given tree is rooted at some specific vertex. For a rooted tree T, we denote the subtree rooted at vertex u by T_u. For $u \in V(T)$, let $p(u)$ be the parent of u and $C(u)$ be the set of children of u. The *height* $h(u)$ of u is defined to be the number of edges of an unweighted longest path among all paths from vertex u to all leaves in T_u. We also define the height of T_u as $h(u)$. Let $e = (u, p(u))$ and $T - \{e\} = T_u \cup (T - T_u)$. We call T_u the lower subtree of edge e and $T - T_u$ the upper subtree. We use c_u and c_{-u} to denote the centers of T_u and $T - T_u$, respectively, and call c_u and c_{-u} the lower center and the upper center of e, respectively. Similarly, m_u and m_{-u} stand for the pair of lower and upper medians of e, and z_u and z_{-u} stand for the pair of lower and upper centdians of e. The diameters in T_u and $T - T_u$ are denoted by $\mathcal{D}_u$ and $\mathcal{D}_{-u}$, respectively. The lowest common ancestor of u and $v \in A(T)$ is denoted by $LCA(u, v)$. The leaves of T are denoted by $leaf(T)$, which contains the vertices with height zero.

3 A linear time algorithm for the 2-radius problem

Let T be the input tree and $\mathcal{D} = \mathcal{P}[x_1, x_k] = (x_1, x_2, ..., x_k)$ be a diameter of T. As mentioned in Section 2, the partition (U_1, U_2) satisfies $T[U_1] \cup T[U_2] = T - \{e\}$ for some $e \in E(T)$. Thus, to find an optimal partition we compute for each edge the sum of radii of the corresponding partition and then find the optimal one. The radius of each part in the partition with respect to the removal of an edge e can be determined via identifying the locations of the lower and upper centers of e. Our algorithm works as follows: First, we compute $\mathcal{D}$ and root T at x_1. Second, we locate all lower centers and then all upper centers. Finally, we find the pair of lower and upper centers whose sum of eccentricities is minimum, and the corresponding partition is the solution.

Both finding a diameter of a tree and transforming an unrooted tree into a rooted one can be done in $O(n)$ time [26].

All lower centers can be computed inductively on the subtree height. For each vertex u, we append the following values:

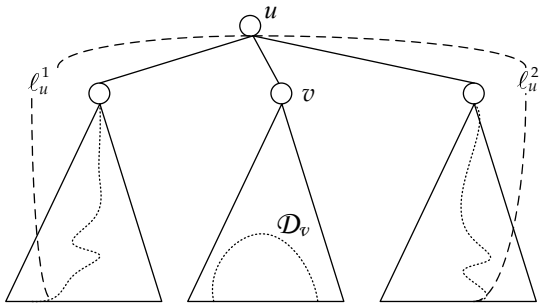

Figure 2: Two possible locations for the diameter of a subtree T_u.

$$\ell_u^1 \equiv \begin{cases} 0, & \text{if } u \text{ is a leaf,} \\ \max_{s \in C(u)}\{d_T(s,u) + \ell_s^1\}, & \text{otherwise.} \end{cases}$$

Let s' be the vertex where $d_T(s',u) + \ell_{s'}^1 = \ell_u^1$.

$$\ell_u^2 \equiv \begin{cases} 0, & \text{if } u \text{ is a leaf or } |C(u)| = 1, \\ \max_{s \in C(u)-\{s'\}}\{d_T(s,u) + \ell_s^1\}, & \text{otherwise,} \end{cases}$$

$$\rho_u \equiv \begin{cases} 0, & \text{if } u \text{ is a leaf,} \\ \max_{s \in C(u)} |\mathcal{D}_s|, & \text{otherwise.} \end{cases}$$

For all $u \in V(G)$, the values ℓ_u^1, ℓ_u^2, ρ_u can be computed while locating the lower centers, and all of them are initialized to be zero. While processing a subtree T_v with $0 \le h(v) \le k$, we compute c_v, record $f_c^{T_v}(c_v)$ and $|\mathcal{D}_v|$, and update $\ell_{p(v)}^1$, $\ell_{p(v)}^2$, and $\rho_{p(v)}$ if $p(v)$ exists. The value $\ell_{p(v)}^2$ is set to be $\ell_{p(v)}^1$ if $d_T(v,p(v)) + \ell_v^1 > \ell_{p(v)}^1$ and to be $d_T(v,p(v)) + \ell_v^1$ if $\ell_{p(v)}^2 < d_T(v,p(v)) + \ell_v^1 \le \ell_{p(v)}^1$. The value $\ell_{p(v)}^1$ is set to be $d_T(v,p(v)) + \ell_v^1$ if $d_T(v,p(v)) + \ell_v^1 > \ell_{p(v)}^1$. While computing the center of T_v, we consider the following two cases (as illustrated in Figure 2): (i) $v \notin \mathcal{D}_v$, and (ii) $v \in \mathcal{D}_v$, which can be identified by comparing ρ_v and $\ell_v^1 + \ell_v^2$. In case (i), $\rho_v > \ell_v^1 + \ell_v^2$, and $\mathcal{D}_v = \mathcal{D}_{v'}$, where $|\mathcal{D}_{v'}| = \rho_v$. Thus $c_v = c_{v'}$, $f_c^{T_v}(c_v) = f_c^{T_{v'}}(c_{v'})$. The values $\ell_{p(v)}^1$, $\ell_{p(v)}^2$, and $\rho_{p(v)}$ are updated accordingly. In case (ii), $\rho_v \le \ell_v^1 + \ell_v^2$, and $\mathcal{D}_v = \mathcal{P}[v,x] \cup \mathcal{P}[v,y]$, where $\mathcal{P}[v,x]$ and $\mathcal{P}[v,y]$ are the corresponding paths with lengths ℓ_v^1 and ℓ_v^2, respectively. Without loss of generality, we assume that $d_T(v,x) \ge d_T(v,y)$, and thus c_v is in $\mathcal{P}[v,x]$. The subtree center c_v can be searched

by accumulating the lengths of edges in the path $\mathcal{P}[c_{v'}, v]$ from $c_{v'}$ toward v, where $v' \in C(v)$ and $x \in V(T_{v'})$. By Property 3, once the accumulation reaches $|\mathcal{D}_v|/2 - f_c^{T_{v'}}(c_{v'})$, the search procedure stops, and c_v is on the edge. The eccentricity of c_v in T_v is $f_c^{T_v}(c_v) = |\mathcal{D}_v|/2$. The reason why c_v is in $\mathcal{P}[v, c_{v'}]$ is because $\mathcal{P}[v', c_{v'}] \subseteq \mathcal{P}[v', x] \subseteq \mathcal{P}[v, x]$, and c_v cannot lie in $\mathcal{P}[c_{v'}, x]$ since otherwise $f_c^{T_v}(c_v) > f_c^{T_v}(c_{v'})$. This procedure can be done in $O(n)$ time since our search always starts from some subtree center toward the subtree root and stop when the center is found. Similar to case (i), the values $\ell^1_{p(v)}$, $\ell^2_{p(v)}$, and $\rho_{p(v)}$ can be updated accordingly.

Now let us discuss the procedure for finding all upper centers. It can be easily shown that for those removal edges not in $\mathcal{D}$, the corresponding upper centers are the center of T. For those edges removed from $\mathcal{D}$, we compute all corresponding upper centers by a top-down approach. This problem is also solved in [25]. For completeness, we show how the approach works in the following. Our goal is to compute the center of $T - T_{x_i}$ for $1 < i \leq k$. First, we compute $\max_{v \in V(T_{x_i} - T_{x_{i+1}})} d_T(x_i, v)$ and $d_T(x_1, x_i)$ for $1 \leq i < k$. Both of these can be done in linear time by traversing $T_{x_i} - T_{x_{i+1}}$ from x_i for all $1 \leq i < k$ and traversing T from x_1, respectively. Second, we compute the centers of the upper subtrees inductively on i, the index of the vertices in $\mathcal{D}$. For each step, we record the center, the radius, and the length of the diameter of the subtree. The basic step is easy since $T - T_{x_2} = x_1$. The center is x_1, the radius is zero, and the length of the diameter is zero. In order to locate the center of $T - T_{x_{i+1}}$, we first compute $\mathcal{D}_{-x_{i+1}}$. Because x_1 is an end vertex of $\mathcal{D}$ and $x_i \in \mathcal{D}$, one can see that x_1 is the farthest vertex from x_i in $T - T_{x_{i+1}}$. Otherwise, $\mathcal{D}$ can be extended to a longer path by substituting $\mathcal{P}[x_i, x_1]$ with the path from x_i to the farthest vertex from x_i in $T - T_{x_{i+1}}$. By Property 2, the center of $T - T_{x_{i+1}}$ is in $\mathcal{P}[x_i, x_1]$, and there are two possible cases (see Figure 3): (i) $\mathcal{D}_{-x_{i+1}} \subseteq T - T_{x_i}$, and (ii) $\mathcal{D}_{-x_{i+1}} \not\subseteq T - T_{x_i}$. If $|\mathcal{D}_{-x_i}| \geq d_T(x_i, x_1) + \max_{v \in V(T_{x_i} - T_{x_{i+1}})} d_T(x_i, v)$, it is case (i). In this case, $\mathcal{D}_{-x_{i+1}} = \mathcal{D}_{-x_i}$, $c_{-x_{i+1}} = c_{-x_i}$, and $f_c^{T - T_{x_{i+1}}}(c_{-x_{i+1}}) = f_c^{T - T_{x_i}}(c_{-x_i})$. If $|\mathcal{D}_{-x_i}| < d_T(x_i, x_1) + \max_{v \in V(T_{x_i} - T_{x_{i+1}})} d_T(x_i, v)$, it is case (ii). In this case, $\mathcal{D}_{-x_{i+1}} = \mathcal{P}[x_i, x_1] \cup \mathcal{P}[x_i, u]$, where $d_T(x_i, u) = \max_{v \in V(T_{x_i} - T_{x_{i+1}})} d_T(x_i, v)$. By Property 2, it can be shown that $c_{-x_{i+1}}$ is in $\mathcal{P}[x_1, x_i]$. Moreover, $c_{-x_{i+1}} \in V(\mathcal{P}[x_i, c_{-x_i}])$ since otherwise $f_c^{T - T_{x_{i+1}}}(c_{-x_{i+1}}) > f_c^{T - T_{x_{i+1}}}(c_{-x_i})$. Thus, $c_{-x_{i+1}}$ can be determined by accumulating the lengths of the edges in $\mathcal{P}[x_i, c_{-x_i}]$ from c_{-x_i} until it reaches $|\mathcal{D}_{-x_{i+1}}|/2 - d_T(x_1, c_{-x_i})$ (Property 3). The eccentricity is $f_c^{T - T_{x_{i+1}}}(c_{-x_{i+1}}) = |\mathcal{D}_{-x_{i+1}}|/2$. All upper centers can be found in linear time

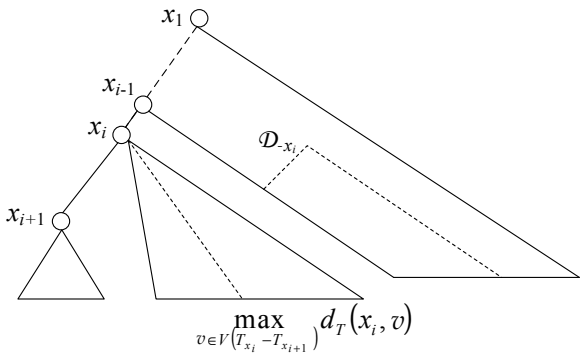

Figure 3: Two possible locations for $\mathcal{D}_{-x_{i+1}}$.

since for the deleted edge $e \notin \mathcal{D}$, the corresponding upper centers are the center of T, and for $e \in \mathcal{D}$, the corresponding upper centers are always searched toward x_k from the previous computed upper center. Thus the total time for computing all upper centers is $O(n)$.

It takes $O(n)$ time to find the pair of lower and upper centers of an edge whose sum of eccentricities is minimum since there are n pairs of lower and upper centers. From the above analysis, one can see that each step takes linear time. Thus we have the following theorem.

Theorem 4: The 2-radius problem on a tree can be solved in linear time.

4 An $O(n \log n)$-time algorithm for the 2-radiian problem

In the following, we root the input tree T at the median m. Our algorithm for the 2-radiian problem is like the "link deletion" method [7] and works as follows. First, in a preprocessing stage, we construct the data structure for querying the lowest common ancestor of two points, evaluate for $a \in V(T)$ the values of $d_T(a, m), f_m^T(a), f_m^{T_a}(a), w(T)$, and $w(T_a)$, and identify the end points of all candidate paths. Second, we find all lower centdians and then all upper centdians. Finally we determine the 2-radiian of T by finding the pair of lower and upper centdians with minimum sum of centdian values.

In the preprocessing stage, we construct a data structure to answer the LCA query (the query for the lowest common ancestor of two given vertices) in constant time, and this can be done in linear time [3, 13]. Moreover, we compute the values $d_T(a, m), f_m^T(a), f_m^{T_a}(a), w(T)$, and $w(T_a)$,

which can be done in linear time for all $a \in V(T)$ [7]. All upper centers, upper medians, lower centers, and lower medians can be found in $O(n \log n)$ time by using the method for maintaining centers and medians in dynamic trees [1]. The pairs of medians can also be found by the algorithm for solving 2-median problem on trees in $O(n \log n)$ time [7]. For all pairs of centers, we can also use the algorithm in Section 3. For convenience, if there is no vertex at some upper or lower center location, we insert an auxiliary vertex with weight zero to such a location. Thus we can assume that all lower and upper centers lie on vertices. This process can be done in linear time by the method in Section 3. Via this process, the end points of all candidate paths and the possible locations of all lower and upper centdians are vertices [8, 21].

To compute all lower centdians, we use binary search on the candidate path for each lower subtree. However, if every candidate path is stored separately in an array, the space would be $O(n^2)$. Lemma 5 overcomes this difficulty.

Lemma 5: Each component of $\bigcup_{u \in V(T)} \mathcal{P}[c_u, LCA(c_u, m_u)]$ and $\bigcup_{u \in V(T)} \mathcal{P}[m_u, LCA(c_u, m_u)]$ is a path.

Proof: To prove that each component of $\bigcup_{u \in V(T)} \mathcal{P}[c_u, LCA(c_u, m_u)]$ is a path, we claim that, in $\bigcup_{u \in V(T)} \mathcal{P}[c_u, LCA(c_u, m_u)]$, there is no vertex with degree larger than two. Suppose to the contrary that there is a vertex with degree larger than two. Let v be such a vertex with minimum height in T. There must be two vertices v' and v'' which satisfy $h(v') \geq h(v)$, $h(v'') \geq h(v)$, and $v \in \mathcal{P}[c_{v'}, LCA(c_{v'}, m_{v'})] \cap \mathcal{P}[c_{v''}, LCA(c_{v''}, m_{v''})]$. Without loss of generality, we assume that $h(v') \geq h(v'')$. Similar to the argument in Section 3, if $c_{v'} \in T_{v''}$, then $c_{v'} \in \mathcal{P}[c_{v''}, v'']$, which leads to a contradiction.

For $\bigcup_{u \in V(T)} \mathcal{P}[m_u, LCA(c_u, m_u)]$, it has been shown in [7, 17] that $m_{v'} \in \mathcal{P}[m_{v''}, v'']$, where $h(v') \geq h(v'')$ and $m_{v'} \in v(T_{v'})$. Therefore, a similar argument holds, and the lemma follows. □

As a result, we store each path of $\bigcup_{u \in V(T)} \mathcal{P}[c_u, LCA(c_u, m_u)]$ and $\bigcup_{u \in V(T)} \mathcal{P}[m_u, LCA(c_u, m_u)]$ in an array. When searching for the centdian of T_u for some $u \in V(T)$, we identify the paths which contain $\{c_u, LCA(c_u, m_u)\}$ and $\{m_u, LCA(c_u, m_u)\}$, respectively, and apply binary search on them. The searching process works as follows. Let $A[r..s]$ be the array where we search for z_u. When

$r = s$, the element $A[r]$ corresponds to the vertex with minimum centdian value in A, and the searching process stops. According to the convexity of the centdian function, if $f_\lambda^{T_u}(A[\lfloor \frac{r+s}{2} \rfloor]) \leq f_\lambda^{T_u}(A[\lfloor \frac{r+s}{2} \rfloor + 1])$, the subarray $A[r..\lfloor \frac{r+s}{2} \rfloor]$ is searched recursively. Otherwise, $A[\lfloor \frac{r+s}{2} \rfloor + 1..s]$ is searched. After both arrays, which contain $\{c_u, LCA(c_u, m_u)\}$ and $\{m_u, LCA(c_u, m_u)\}$, respectively, are searched, we compare the centdian values of the resulting elements and choose the one with minimum centdian value. To determine $f_\lambda^{T_x}(y)$ for $x, y \in V(T)$ and $y \in V(T_x)$, we give the following formulae. The median function $f_m^{T_x}(y)$ can be obtained by the formula

$$f_m^{T_x}(y) = f_m^T(y) - \left(f_m^T(x) - f_m^{T_x}(x)\right) - (w(T) - w(T_x)) \cdot d_T(x, y), \tag{1}$$

where $d_T(x, y)$ can be computed by $d_T(y, m) - d_T(x, m)$. To compute $f_c^{T_x}(y)$, by Property 3, we have

$$f_c^{T_x}(y) = d_T(m, c_x) + d_T(m, y) - d_T(m, LCA(c_x, y)) + f_c^{T_x}(c_x), \tag{2}$$

where each term inside is pre-computed. By formulae (1) and (2), the value $f_\lambda^{T_x}(y)$ can be answered in constant time. Thus, the time complexity for computing all the lower centdians is $O(n \log n)$.

Our method for finding all upper centdians works as follows. First, we reduce the size of the candidate set which contains all possible upper centdian locations. Second, we decompose the subgraph induced by the vertices in the reduced candidate set so that it can be stored and accessed efficiently. When searching for an upper centdian, we identify the candidate path and apply binary search on it.

The size-reduced candidate set is the vertex set of a subtree T', which contains the candidate paths of all upper subtrees. Let $\mathcal{P}[\alpha_1, \alpha_2]$ be a diameter of T, and without loss of generality, we assume that $d_T(m, \alpha_1) \geq d_T(m, \alpha_2)$. The subtree T' is defined as $\mathcal{P}[m, m_1] \cup \mathcal{P}[m, m_2] \cup \mathcal{P}[m, \alpha_1] \cup T[X]$, where m_1 and m_2 are the medians of the heaviest and second heaviest subtrees of $T - \{m\}$, and $X = \{V(\mathcal{P}[v, p(x)]) : x \in V(\mathcal{P}[m, \alpha_1]) - \{m\}, v \in V(T_{p(x)} - T_x),$ and $d_T(v, p(x)) = \max_{s \in V(T_{p(x)} - T_x)} d_T(s, p(x))\}$ (cf. Figure 4). The subtree T' is well defined since the upper medians are in $\mathcal{P}[m, m_1] \cup \mathcal{P}[m, m_2]$ [7], the upper centers are in $T[X]$ by Property 2, and $\mathcal{P}[c_{-x}, m_{-x}] \subseteq \mathcal{P}[m, c_{-x}] \cup \mathcal{P}[m, m_{-x}] \subseteq T'$ for all $x \in V(T) - leaf(T)$. The construction of T' takes $O(n)$ time since m_1 and m_2 can be computed in linear time [7], and the vertices in $T[X]$ can also be computed

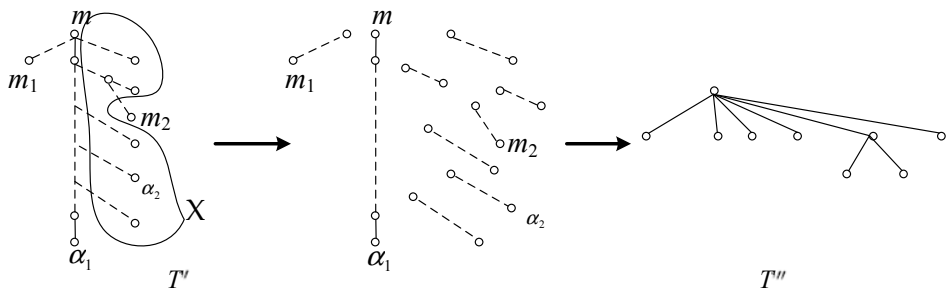

Figure 4: The decomposition of T' and the auxiliary tree T''.

in linear time by traversing $T_{p(x)} - T_x$ from $p(x)$ and backtracking the path from the farthest vertex from $p(x)$ in $T_{p(x)} - T_x$ for all $x \in V(\mathcal{P}[m, \alpha_1]) - \{m\}$.

To store and access T' efficiently, we decompose T' into a set of paths, and for each path we store its vertices in an array. The decomposition is formed by splitting the vertices v with degree greater than two into $deg_{T'}(v)$ vertices, and uniting the paths whose two end vertices are in $\mathcal{P}[m, \alpha_1]$, where $deg_{T'}(v)$ denotes the degree of v in T' (see Figure 4).

For an edge $(x, p(x)) \in E(T)$, before applying binary search on $\mathcal{P}[c_{-x}, m_{-x}]$, we need to identify the arrays which compose $\mathcal{P}[c_{-x}, m_{-x}]$. For convenience, we construct an auxiliary tree T'' whose vertex set corresponds to the paths in the decomposition, and two vertices of T'' are adjacent if there is a common vertex $v \in V(T)$ in both of the corresponding paths. Let the vertex which corresponds to $\mathcal{P}[m, \alpha_1]$ be the root of T''. The set of arrays $\mathcal{A}_x$ which covers $\mathcal{P}[c_{-x}, m_{-x}]$ can be obtained by backtracking T'' from the vertices, which correspond to the arrays containing c_{-x} and m_{-x}, to the vertex, which corresponds to the array containing $LCA(c_{-x}, m_{-x})$. We shall show, in Lemma 6, that the set of arrays $\mathcal{A}_x$ can be identified in constant time. One can see that some elements of the arrays in $\mathcal{A}_x$ do not correspond to the vertices in $\mathcal{P}[c_{-x}, m_{-x}]$. These elements can be removed by the following procedure (see Figure 5). Let $\mathcal{A}_x = \{A_1, A_2, ..., A_k\}$ with $c_{-x} \in A_1$, $m_{-x} \in A_k$, and $|A_i \cap A_{i+1}| = 1$ for $1 \leq i < k$. An element y of A_i is not in $\mathcal{P}[c_{-x}, m_{-x}]$ if and only if $y \in A_i[1..l_i - 1] \cup A_i[r_i + 1..s_i]$, where $A_{i-1} \cap A_i = A_i[l_i]$, $A_i \cap A_{i+1} = A_i[r_i]$, and $s_i = |A_i|$. Otherwise, $\mathcal{P}[c_{-x}, m_{-x}]$ would not be a path. The elements $A_i[l_i]$ and $A_i[r_i]$ are recorded while backtracking T''. Let $\mathcal{B}_x = \{B_1, B_2, ..., B_k\}$, where $B_i = A_i[l_i..r_i]$ for $1 \leq i < k$ (see Figure 5). Again, by Lemma 6, identifying $\mathcal{B}_x$ can be done in constant time.

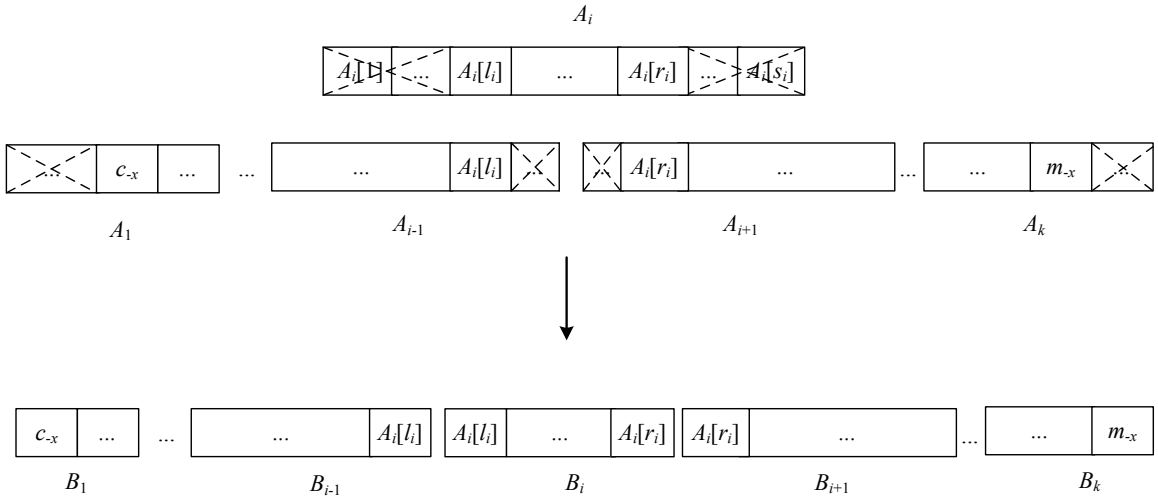

Figure 5: The set of arrays $\mathcal{A}_x$ and $\mathcal{B}_x$.

Lemma 6: For $x \in V(T) - \{m\}$, we have $|\mathcal{A}_x| = |\mathcal{B}_x| \leq 5$.

Proof: The equality holds because B_i is a continuous part of A_i, for $B_i \in \mathcal{B}_x$ and $A_i \in \mathcal{A}_x$. Since $\mathcal{A}_x$ corresponds to a path in T'', we show in the following that the length of a longest path in T'' is no more than four. In T', we claim that except the vertices in $\mathcal{P}[m, \alpha_1]$, there are at most two vertices with degree larger than two. From the fact that the intersection of a path and a tree increases the degrees of at most two vertices in the tree, the claim holds since T' is formed by uniting $\mathcal{P}[m, m_1]$ and $\mathcal{P}[m, m_2]$ with the tree $\mathcal{P}[m, \alpha_1] \cup T[X]$. Therefore, the length of a longest path in T'' is no more than four, and the inequality holds. □

When applying binary search on $\mathcal{P}[c_{-x}, m_{-x}]$, we need to compute the $\left(\lfloor \frac{\sum_{i=1}^{k} |B_i| - (k-1)}{2} \rfloor\right)$-th element first. This can be done by pre-computing the size of each subarray B_i for $1 \leq i \leq k$, and then finding the number k' such that

$$\sum_{i=1}^{k'-1} (|B_i| - 1) < \lfloor \frac{\sum_{i=1}^{k} |B_i| - (k-1)}{2} \rfloor \leq \sum_{i=1}^{k'} (|B_i| - 1).$$

By Lemma 6, one is able to see that k' can be found in constant time, and the searched element is $B_{k'}[s]$, where $s = \left(\lfloor \frac{\sum_{i=1}^{k} |B_i| - (k-1)}{2} \rfloor - \sum_{i=1}^{k'-1} (|B_i| - 1) \right)$. If $B_{k'}[s]$ is the centdian, the procedure

stops and records the centdian and its centdian value. If the centdian is on the left of $B_{k'}[s]$, then we apply the above procedure recursively on the arrays $B_1, B_2, ..., B_{k'}[1..s-1]$. Otherwise, the procedure runs recursively on $B_{k'}[s+1..|B_{k'}|], B_{k'+1}, ..., B_k$. The time to search an upper centdian is $O(\log n)$ multiplied by the time to determine whether a searched vertex is a centdian.

For a given vertex y on $\mathcal{P}[c_{-x}, m_{-x}]$, to determine $f_\lambda^{T-T_{-x}}(y)$, we give the following formulae. The median function $f_m^{T-T_x}(y)$ can be obtained by the formula

$$f_m^{T-T_x}(y) = f_m^T(y) - f_m^{T_x}(x) - w(T_x) \cdot d_T(x, y),$$

where each term in this formula can be answered in constant time after an $O(n)$-time preprocessing [7]. Similar to formula (3), the center function is computed by

$$f_c^{T-T_x}(y) = d_T(m, c_{-x}) + d_T(m, y) - d_T(m, LCA(c_{-x}, y)) + f_c^{T-T_x}(c_{-x}).$$

According to these two formulae, the centdian function $f_\lambda^{T-T_x}(y)$ can be answered in constant time. Thus, the total time for computing the upper centdians is $O(n \log n)$.

To determine the optimal partition, we compute, for each edge, the sum of centdian values of the pair of lower and upper centdians and find the minimum. The corresponding partition is the 2-radiian of T. This can be done in linear time since there are n pairs of lower and upper centdians. We conclude this section with the following theorem.

Theorem 7: The 2-radiian problem on a tree can be solved in $O(n \log n)$ time.

5 Concluding remarks

In this paper, we consider two facility-centric facility location problems, the 2-radius and the 2-radiian problems on trees, and give $O(n)$-time and $O(n \log n)$-time algorithms, respectively. Both algorithms can be applied to the $V(T)/V(T)/2$ case by substituting the continuous centers to discrete ones (centers only on vertices), and when computing the discrete centers and their eccentricities, we use the following property instead of Property 3.

Property 8: [26] Let $\mathcal{D}$ be a diameter of a tree T and c be the continuous center of T. The discrete center is the vertex u, where $c \in e = (u, v)$ for some $e \in E(T)$ and $d_T(c, u) \leq d_T(c, v)$. The eccentricity of u is $|\mathcal{D}|/2 + d_T(c, u)$.

In the 2-radiian problem, people may want to normalize the median function with respect to the number of vertices or the sum of vertex weights in the corresponding part of a partition. Fortunately, our algorithm for the 2-radiian problem can be adopted to resolve this reformulation, and the time complexity remains unchanged. The reasons are that the centdian function remains convex when each vertex weight is divided by the number of vertices or the sum of vertex weights, and that both terms can be obtained in constant time after a linear-time preprocessing [7, 26].

In our objective function, the center function is unweighted. A natural extension is to consider the weighted center function in both objective functions of the radius and the radiian problems. The p-radiian problem on trees for arbitrary p is also an interesting topic to work on in the future.

Acknowledgments. The authors would like to thank the anonymous reviewer, Kuan-Yu Chen, and Hsiao-Fei Liu for improving the presentation of the paper. Hung-Lung Wang and Kun-Mao Chao were supported in part by NSC grants 95-2221-E-002-078, 95-2221-E-002-126-MY3, and 96-2221-E-002-034 from the National Science Council, Taiwan.

References

[1] S. Alstrup, J. Holm, and M. Thorup, Maintaining center and median in dynamic trees, *In Proceedings of the 7th Scandinavian Workshop on Algorithm Theory* (*Lecture Notes in Computer Science 1851*), pp. 46–56, 2000.

[2] B. Ben-Moshe, B. Bhattacharya, and Q. Shi, Efficient algorithms for the weighted 2-center problem in a cactus graph, *In Proceedings of the 16th International Symposium on Algorithms and Computation* (*Lecture Notes in Computer Science 3827*), pp. 553–562, 2005.

[3] M. A. Bender and M. Farach-Colton, The LCA problem revisited, *In Proceedings of the 4th Latin American Symposium on Theoretical Informatics* (*Lecture Notes in Computer Science 1776*), pp. 88–94, 2000.

[4] D. Bilò, J. Derungs, L. Gualà, G. Proietti, and P. Widmayer, Locating facilities on a network to minimize their average service radius, *In Proceedings of the 18th International Symposium on Algorithms and Computation* (*Lecture Notes in Computer Science 4835*), pp. 587–598, 2007.

[5] E. Erkut, R. L. Francis, T. J. Lowe, and A. Tamir, Equivalent Mathematical Programming Formulations of Monotonic Tree Network Location, *Operations Research*, 37:3, pp. 447–461, 1989.

[6] G. N. Frederickson, Parametric search and locating supply centers in trees, *In Proceedings of the 2nd Workshop on Algorithms and Data Structures* (*Lecture Notes in Computer Science 519*), pp. 299–319, 1991.

[7] B. Gavish and S. Shridhar, Computing the 2-median on tree networks in $O(n \log n)$ time, *Networks*, 26, pp. 305–317, 1995.

[8] J. Halpern, The location of a center-median convex combination on an undirected tree, *Journal of Regional Science*, 16: 2, pp. 237–245, 1976.

[9] J. Halpern, Finding minimal center-median convex combination (cent-dian) of a graph, *Management Science*, 24: 5, pp. 535–544, 1978.

[10] J. Halpern, Duality in the cent-dian of a graph, *Operations Research*, 28, pp. 722–735, 1980.

[11] G. Y. Handlar, Medi-centers of a tree, *Transportation Science*, 19: 3, pp. 246–260, 1985.

[12] F. Harary, *Graph Theroy*, Addison-Wesley, Reading, Mass, 1969.

[13] D. Harel and R. E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM Journal on Computing*, 13, pp. 338–355, 1984.

[14] M. Jeger and O. Kariv, Algorithms for finding p-centers on a weighted tree (for relatively small p), *Networks*, 15, pp. 381–389, 1985.

[15] O. Kariv and S. L. Hakimi, An algorithmic approach to network location problems, Part I: The p-centers, *SIAM Journal on Applied Mathematics*, 37: 2, pp. 441–461, 1979.

[16] O. Kariv and S. L. Hakimi, An algorithmic approach to network location problems, Part II: The p-medians, *SIAM Journal on Applied Mathematics*, 37: 3, pp. 539–560, 1979.

[17] S. C. Ku, C. J. Lu, B. F. Wang, and T. C. Lin, Efficient algorithms for two generalized 2-median problems on trees, *In Proceedings of the 12th International Symposium on Algorithms and Computation* (*Lecture Notes in Computer Science 2223*), pp. 768–778, 2001.

[18] Y. F. Lan, Y. L. Wang and H. Suzuki, A linear-time algorithm for solving the center problem on weighted cactus graphs, *Information Processing Letters*, 71, pp. 205–212, 1999.

[19] N. Megiddo, A. Tamir, E. Zemel and R. Chandrasekaran, An $O(n \log^2 n)$ algorithms for the kth longest path in a tree with applications to location problems, *SIAM Journal on Computing*, 10: 2, pp. 328–337, 1981.

[20] W. Ogryczak and A. Tamir, Minimizing the sum of the k largest functions in linear time, *Information Processing Letters*, 85, pp. 117–122, 2003.

[21] D. Pérez-Brito, J. A. Moreno-Pérez, and I. Rodríguez-Martín, The 2-facility centdian network problem, *Location Science*, 6, pp. 369–381, 1998.

[22] G. Proietti and P. Widmayer, Partitioning the nodes of a graph to minimize the sum of subgraph radii, *In Proceedings of the 17th International Symposium on Algorithms and Computation* (*Lecture Notes in Computer Science 4288*), pp. 578–587, 2006.

[23] A. Tamir, D. Pérez-Brito, and J. A. Moreno-Pérez, A polynomial algorithm for the p-centdian problem on a tree, *Networks*, 32, pp. 255–262, 1998.

[24] A. Tamir, J. Puerto, and D. Pérez-Brito, The centdian subtree on tree networks, *Discrete Applied Mathematics*, 118, pp. 263–278, 2002.

[25] H. L. Wang, B. Y. Wu, and K. M. Chao, The backup 2-center and backup 2-median problems on trees, submitted to *Networks*.

[26] B. Y. Wu and K. M. Chao, *Spanning Trees and Optimization Problems*, Chapman and Hall/CRC Press, USA, 2004.