# Efficient Algorithms for Two Generalized 2-Median Problems on Trees*

Shan-Chyun Ku[1], Chi-Jen Lu[2], and Biing-Feng Wang[3]

[1] *Core Technology Department, Faraday Technology Corporation*
*Hsinchu, Taiwan, Republic of China*

[2] *Institute of Information Science, Academia Sinica,*
*Taipei, Taiwan, Republic of China*

[3] *Department of Computer Science, National Tsing Hua University*
*Hsinchu, Taiwan 30043, Republic of China*

**Running Head:** GENERALIZED 2-MEDIAN

**Correspondence Address:**

Prof. Biing-Feng Wang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30043, TAIWAN
Phone: 886-3-5742805
Fax: 886-3-5723694
E-mail: bfwang@cs.nthu.edu.tw

* This research is supported by the National Science Council of the Republic of China under grant NSC-90-2213-E-007-071. A preliminary version of this paper was presented at *the 12th Annual International Symposium on Algorithms and Computation*, 2001.

# Efficient Algorithms for Two Generalized 2-Median Problems on Trees

Shan-Chyun Ku[1], Chi-Jen Lu[2], and Biing-Feng Wang[3]

[1] *Core Technology Department, Faraday Technology Corporation*
*Hsinchu, Taiwan, Republic of China*

[2] *Institute of Information Science, Academia Sinica,*
*Taipei, Taiwan, Republic of China*

[3] *Department of Computer Science, National Tsing Hua University*
*Hsinchu, Taiwan 30043, Republic of China*

## Abstract

The $p$-median problem on a tree $T$ is to find a set $S$ of $p$ vertices on $T$ that minimize the sum of distances from $T$'s vertices to $S$. For this problem, Tamir [14] had an $O(pn^2)$-time algorithm, while Gavish and Sridhar [6] had an $O(n\log n)$-time algorithm for the case of $p$=2. In this paper, we study two generalizations of the 2-median problem, which are obtained by imposing constraints on a 2-median: one is to limit their eccentricity while the other is to limit their distance. We solve both generalizations in $O(n\log n)$ time, improving the previous ones from $O(n^2)$. We also study cases when linear time algorithms exist for the 2-median problem and the two generalizations. For example, we solve all three in linear time when edge lengths and vertex weights are all polynomially bounded integers. Finally, we consider the relaxation of the two generalized problems by allowing 2-medians on any position of edges, instead of just on vertices, and we give $O(n\log n)$-time algorithms for them.

**Keywords:** trees, $p$-median, eccentricity, group median, network location theory.

# 1. Introduction

Optimally locating a set of facilities on a network is an important problem in the fields of transportation and communication [9, 10, 13]. One classical problem is *the p-median problem*: Given a graph $G=(V, E)$, in which each edge has a nonnegative length, and a number $p$, find a set $S$ of $p$ vertices to minimize some distance-sum function on $S$. This models the scenario of placing $p$ facilities to minimize the average access cost from the network. Let $d(v, u)$ be the distance between vertices $v$ and $u$ and let $d(v, S)=\min_{u\in S} d(v, u)$. Kariv and Hakimi [11] considered the distance-sum function $Sum_G(S)=\Sigma_{v\in V} d(v, S)$ and showed that the problem is NP-hard. So one may only hope to find efficient algorithms for special classes of graphs. For trees, they provided in that paper an $O(p^2n^2)$-time algorithm, while Tamir [14] had an $O(pn^2)$-time algorithm with respect to a generalized distance-sum function. From now on, trees will be the only subject of our discussion and all our results are stated with respect to trees. Some attention has been given to the cases of $p$=1 and $p$=2. Goldman [7] had a linear time algorithm for $p$=1. Gavish and Sridhar [6] had an $O(n\log n)$-time algorithm for $p$=2 using the following generalized distance-sum function:

$$Sum_T(S) = \Sigma_{v\in V} d(v, S)\times w(v),$$

where $T=(V, E)$ is a tree in which every vertex $v\in V$ has a nonnegative weight $w(v)$. Auletta *et al.* [1] presented another algorithm for the 2-median problem. The idea of the algorithms in [1, 6] are the same. They are different implementations of a strategy called *the link-deletion method* [12]. Without giving a formal analysis, Auletta *et al.* asserted that the running time of their algorithm is $O(n)$. In fact, their linear-time assertion is incorrect. As we show in Appendix B that any implementation of the link-deletion method has an $\Omega(n\log n)$ lower bound, even when every edge has unit length.

Tamir *et al.* [15] introduced a generalization of the $p$-median problem by limiting the eccentricity of $p$-medians and gave an $O(pn^2)$-time algorithm for it. The *eccentricity* of a subset $S\subseteq V$ is defined as $Ecc_T(S)=\max_{v\in V} d(v, S)$. They also studied the generalization with relaxation that allows $p$-medians on any position of edges, instead of just on vertices. They solved the relaxed version in $O(pn^3)$ time. From some practical point of view, Wang *et al*. [18] introduced another generalization of the $p$-median problem by limiting the longest distance between any pair of the vertices selected as a $p$-median. They gave an $O(n^2)$-time algorithm for the case of $p$=2. To the authors' knowledge, no algorithms had been proposed for general $p$. A simple polynomial time solution is as follows. Let $l$ be the given limitation on the longest distance. Tamir's algorithm in [15] for the classical $p$-median problem allows each vertex to have a setup cost. Therefore, given a subtree $X$ of $T$, Tamir's algorithm can be applied to select the best $p$ vertices within $X$ by letting the setup cost of every vertex outside $X$ be $\infty$. For any pair of $a$, $b\in V$ with $d(a, b)\leq l$, we define $X_{ab}$ as the subtree containing all vertices $v\in V$ having $d(v, c)\leq l/2$, where $c$ is the midpoint of the path from $a$ to $b$. There are at most $O(n^2)$ such subtrees. Clearly, at least one of them covers a $p$-median. Thus, by performing Tamir's algorithm once for every $X_{ab}$, we can solve the generalization in polynomial time for any $p$. We remark that different constrained 2-median problems were studied in [2, 3].

The cases of $p$=2 of the above two generalized problems are the focus of this paper. We solve both in $O(n\log n)$ time, matching even the fastest algorithm currently known for the 2-median problem. We can even do better for some natural cases. In linear time, we can solve the 2-median problem with constant edge lengths or polynomial vertex weights, the first generalization with polynomial vertex weights, and the second generalization with polynomial vertex weights and polynomial edge lengths. Finally, we consider the relaxation that allows 2-medians on any position of edges, instead of just on vertices. We give $O(n\log n)$-time algorithms for the relaxed version of the two generalized problems.

Along the way, we solve a problem, named *the tree marker problem*, in $O(n\log n)$ time, which may be of independent interest. *Wang et al.* [18] solved a special case of the tree marker problem, called *the tree bisector problem*, in $O(n\log n)$ time and gave an $O(n)$ time algorithm for the case in which all edges have unit length. They asked the following question in their paper: Is it possible to solve the problem in $o(n\log n)$ time when all edge lengths are polynomially bounded integers? In this paper, we answer the question by solving in linear time the corresponding case of the tree mark problem. The tree marker problem arises several times in our approaches to the 2-median problems. Besides, it can be used to speedup one of Gupta and Punnen's group median algorithms in [8]. We elaborate on this in Appendix C.

The rest of this paper is organized as follows. Notation and preliminaries are given in the next section. We solve the generalization with eccentricity constraint in Section 3, and the generalization with distance constraint in Section 4. In Section 5, we study cases when linear time algorithms exist. We solve the relaxed problems in Section 6. Finally, future work is discussed in Section 7.

## 2. Notation and Preliminaries

Given a tree $T$, let $V(T)$ denote its vertex set and let $E(T)$ denote its edge set. Each edge $e$ has a nonnegative length $d(e)$ and each vertex $v$ has a nonnegative weight $w(v)$. Define the weight of a subtree $H$ as $w(H) = \sum_{v\in V(H)} w(v)$. For two vertices $u$ and $v$, let $path(u, v)$ denote the path between them, let their distance $d(u, v)$ be the length of this path, and let $LCA(u, v)$ denote their least common ancestor when $T$ is rooted.

We will use Gavish and Sridhar's distance-sum function defined in the previous section. For notational convenience, we write $Sum_T(v)$ for $Sum_T(\{v\})$ and $Sum_T(u, v)$ for $Sum_T(\{u, v\})$. A set $S$ of $p$ vertices is called a *p-median* of $T$ if $Sum_T(S) \leq Sum_T(S')$ for any $S'$ of $p$ vertices. A *median* of a tree is just its 1-median.

Consider a rooted tree $T$ and a vertex $v$. An *ancestor* of $v$ is a vertex on the path from $v$ to the root, so $v$ is considered an ancestor of itself. Let $par(v)$ denote $v$'s parent in

$T$. Let $T_v$ denote the subtree rooted at $v$, and define

$$\delta_T(v)=w(T_v)\text{-}w(T\backslash T_v)$$

for the weight difference between $T_v$ and its complement. For a child $u$ of $v$, $T_u$ is called a subtree of $v$. The following states an important property of medians on trees.

**Lemma 2.1** On a rooted tree $T$, the vertex $v$ with the smallest $\delta_T(v)>0$ is a median.

**Proof:** Let $v$ be the vertex with the smallest $\delta_T(v)>0$. Imagine moving $v$ to a vertex $r\neq v$. If $r$ is not on $T_v$, vertices of $T_v$ have their distances increased by $d(r, v)$ while others have distances decreased by at most $d(r, v)$, and the distance-sum increases by at least $\delta_T(v)d(r, v)>0$. Otherwise, $r$ is on a subtree $T_u$ of $v$, and vertices outside of $T_u$ have distances increased by $d(r, v)$ while others have distances decreased by at most $d(r, v)$. The distance-sum now increases by at least $-\delta_T(u)d(r, v)$, which is nonnegative as $\delta_T(u)<\delta_T(v)$ and $\delta_T(v)$ is the smallest positive one. Moving $v$ elsewhere does not decrease the distance-sum, so $v$ is a median.

A rooted tree $T$ may have more than one median but it has a unique vertex $v$ with the smallest $\delta_T(v)>0$, and we call $v$ the *median*, denoted as $m(T)$. This lemma basically says that the median gives the most balanced partition of the tree into two parts. We can find $m(T)$ by starting from the root and going to the heaviest subtree each time, as vertices elsewhere have negative $\delta_T$ values. If a vertex $v$ has at least two heaviest subtrees, then all its children have negative $\delta_T$ values, so we don't need to go down further, and $v$ is the median. Let $P(T)$ denote such a path on $T$. Clearly, $m(T)$ can be found this way in linear time.

Next, consider the 2-median problem on trees. We will root $T$ at $m=m(T)$ throughout the paper unless mentioned otherwise. The idea is that with $(m_1, m_2)$ being a 2-median of a tree $T$, those vertices closer to $m_1$ and those others divide $T$ into two subtrees with $m_1$ and $m_2$ being their respective 1-medians. Let $X(e)$ and $Y(e)$ denote the two resulting subtrees when edge $e$ is deleted, with $X(e)$ being the one containing $m$. This motivates the following useful strategy called *the link-deletion method* [6, 12]:

Step 1. Find $m_Y(e)=m(Y(e))$ for every $e\in E(T)$.
Step 2. Find $m_X(e)=m(X(e))$ for every $e\in E(T)$.
Step 3. Output the pair $(m_X(e), m_Y(e))$ with smallest $Sum_{Y(e)}(m_Y(e))+Sum_{X(e)}(m_X(e))$.

Gavish and Sridhar had an $O(n\log n)$-time algorithm, which runs Step 1 and 3 in $O(n)$ time but needs $O(n\log n)$ time for Step 2. Let $T_\alpha$ and $T_\beta$ be $m$'s two heaviest subtrees, with a tie broken arbitrarily. They observed a property that $m_X(e)\in P(T_\beta)$ if $e\in T_\alpha$ and $m_X(e)\in P(T_\alpha)$ otherwise. Let $J_T(e)$ denote that path containing $m_X(e)$.

For vertices $x$ and $y$, define their bisector $BS(x, y)$ as the edge containing the position with equal distance to $x$ and $y$. We have the following properties.

**Property 2.1 [18]** For $e{=}BS(x, y)$ and $x{\in}X(e)$, $Sum_T(x, y){=}Sum_{X(e)}(x){+}Sum_{Y(e)}(y)$.

**Property 2.2 [18]** After an $O(n)$ time preprocessing on $T$, for any $x, y{\in}V$, $Sum_T(x, y)$ can be computed in $O(1)$ time if $BS(x, y)$ is given.

Sometimes it makes sense to consider arbitrary positions, called points, on edges, instead of just vertices. Some definitions above can be easily extended, such as the distance of two points and the distance-sum function. For a point $r{\notin}V(T)$, $\delta_T(r)$ is defined as the weight difference between the two subtrees separated by $r$. This relaxation does not reduce the distance-sum of a 1-median, as moving it to some vertex on that edge still preserves the distance-sum. This is also true for a 2-medain, as the two vertices are the respective 1-medians of $X(e)$ and $Y(e)$ for some edge $e$. However, this is not the case for the generalized 2-median problems.

Later we will encounter some tasks which can be formulated as the following *tree marker problem*. The input is a tree $T$ and a set of $q{=}O(n)$ 3-tuples $(v, l, u)$ with $v, u{\in}V(T)$ and $0{\le}l{\le}d(v, u)$, and for each 3-tuple $(v, l, u)$, we output the point $p$ on $path(v, u)$ with $d(v, p){=}l$. We have the following, which is proved in Appendix A, assuming that sorting $n$ edge lengths takes $t_e$ time.

**Theorem 2.1** The tree marker problem can be solved in $O(n{+}\min\{q\log n, t_e\}){=}O(n\log n)$ time.

## 3. The First Generalization

Recall that the eccentricity of a set $S$ is defined as $Ecc_T(S) = \max_{v\in V(T)} d(v, S)$. For notational convenience, we write $Ecc_T(v)$ for $Ecc_T(\{v\})$ and $Ecc_T(u, v)$ for $Ecc_T(\{u, v\})$. The *diameter* of $T$, denoted as $dm(T)$, is the longest path on $T$. The *center* of $T$, denoted as $c(T)$, is a vertex $u$ such that $Ecc_T(u) \le Ecc_T(v)$ for every vertex $v$ on $T$. It is not hard to verify that $c(T)$ is also a center of $dm(T)$, and $dm(T)$ must end at a farthest leaf of $T$, with $c(T)$ being its ancestor.

The problem to study now is finding two vertices $r_1, r_2{\in}V(T)$ with $Ecc_T(r_1, r_2){\le}l_e$, for a given $l_e$, that minimize $Sum_T(r_1, r_2)$. According to Property 2.1, $Sum_T(r_1, r_2) = Sum_{X(e)}(r_1) + Sum_{Y(e)}(r_2)$ for $e{=}BS(r_1, r_2)$ with $r_1{\in}X(e)$ and $r_2{\in}Y(e)$, and now $Ecc_T(r_1, r_2) = \max\{Ecc_{X(e)}(r_1), Ecc_{Y(e)}(r_2)\}$. So again we can follow the link-deletion method. For an edge $e$, define $r_X(e)$ as a vertex on $X(e)$ with $Ecc_{X(e)}(r_X(e)){\le}l_e$ that minimizes $Sum_{X(e)}(r_X(e))$, and define $r_Y(e)$ on $Y(e)$ accordingly. We will find all such $(r_X(e), r_Y(e))$ and output the pair with the smallest distance-sum. The following lemma tells us where to find $r_Y(e)$ and $r_X(e)$.

**Lemma 3.1** $r_X(e){\in}path(m_X(e), c(X(e)))$ and $r_Y(e){\in}path(m_Y(e), c(Y(e)))$.

This lemma holds as moving away from either path never decreases the eccentricity or the distance-sum. Then $r_X(e)$ is the vertex on $path(m_X(e), c(X(e)))$ nearest to $m_X(e)$ with $Ecc_{X(e)}(r_X(e)) \leq l_e$, and similarly for $r_Y(e)$. We obtain all those $m_X(e)$ and $m_Y(e)$ first. Then, we will find every $dm(Y(e))$ and $c(Y(e))$, or equivalently $dm(T_v)$ and $c(T_v)$ for $e=(v, par(v))$. There are three possibilities for $dm(T_v)$. If $dm(T_v)=dm(T_u)$ for some child $u$ of $v$, then $c(T_v)=c(T_u)$. If $dm(T_v)$ has one end at $v$, then $v$ has only one child $u$ and $dm(T_v)$ must ends at $u$'s farthest leaf, so $c(T_v)\in path(v, c(T_u))$. Otherwise, $dm(T_v)$ is the path through $v$ connecting $v$'s two farthest leaves, then $c(T_v)\in path(v, c(T_u))$ with $T_u$ being $v$'s subtree containing $v$'s farthest leaf. So $c(T_v)$ either stays on or moves up from the center of some subtree $T_u$ of $v$. In a bottom-up way, we can determine which case happens and find $dm(T_v)$ and $c(T_v)$ for every $v$ in linear time.

Next, let's find $r_Y(e)$ from $path(m_Y(e), c(Y(e)))$ for every $e=(v, par(v))$. Let $u$ be the child of $v$ with largest $w(T_u)$. Let $e'=(u, v)$ and $r=r_Y(e')$. If $r$ is not $m_Y(e')$'s ancestor, there must be some leaf $l$ of $T_u$ dragging $r$ from moving up with $d(par(r), l)>l_e$, so $r_Y(e)$ must stay at $r$. Checking this condition for every $e$ in a bottom-up way takes linear time. Otherwise, $r$ is $m_Y(e')$'s ancestor. Since $m_Y(e)$ is $m_Y(e')$'s ancestor, $r_Y(e)$ is the first vertex $r'$ with $Ecc_{Y(e)}(r')\leq l_e$ on the path from $r$ to $m_Y(e)$. If $r_Y(e)$ is an ancestor of $r$, then $r_Y(e)$ moves up from $r=r_Y(e')$. Otherwise, $r_Y(e)$ is not an ancestor of $r$, and thus not an ancestor of $m_Y(e)$. According to the same argument above, $r_Y(e'')=r_Y(e)$ for every $e''$ above $e$, so the path from $LCA(r, c(Y(e)))$ to $c(Y(e)))$ need not be checked again later. Since each vertex is checked only once, the total time is linear.

Finally, let's find $c(X(e))$, and then $r_X(e)\in path(m_X(e), c(X(e)))$, for every $e$. Let $F=path(m, l_1)\cup path(m, l_2)$, where $dm(T)=path(l_1, l_2)$. First, consider those $e\notin F$. Clearly, $dm(X(e))=dm(T)$ and $c(X(e))=c(T)$. If $Ecc_{X(e)}(m_X(e))\leq l_e$, then $r_X(e)=m_X(e)$. Otherwise, $r_X(e)$ is the vertex $r\in J_T(e)\cup path(c(T), m)$ nearest to $m_X(e)$ with $Ecc_T(r)\leq l_e$, which is always one of the at most two (only one when $c(T)\notin J_T(e)$) vertices $r\in J_T(e)\cup path(c(T), m)$ farthest from $c(T)$ with $Ecc_T(r)\leq l_e$. Finding all such $r_X(e)$ takes linear time. Next we consider those $e\in F$. We can change $T$'s root to one end of $dm(T)$ so that each $X(e)$ appears as a subtree $T_v$ with $e=(v, par(v))$. Then all such $r_X(e)$ can be found using the same bottom-up way as that for $r_Y(e)$. Hence, we have the following theorem, assuming that finding all $m_X(e)$ takes $t_x$ time.

**Theorem 3.1** The first generalization can be solved in $O(n+t_x)=O(n\log n)$ time.

## 4. The Second Generalization

The problem studied in this section is to find vertices $r_1$ and $r_2$ on $T$ with $d(r_1, r_2)\leq l_d$, for a given $l_d$, that minimize $Sum_T(r_1, r_2)$. We will follow the framework of the link-deletion method. For an edge $e$, let $r_X(e)$ and $r_Y(e)$ denote the respective vertices on $X(e)$ and $Y(e)$ with $d(r_X(e), r_Y(e))\leq l_d$ that achieve the smallest distance-sum. Root $T$ at $m=m(T)$.

The following lemma helps cut down the search space for $r_X(e)$ and $r_Y(e)$.

**Lemma 4.1** For $e=(v, par(v))$, $r_Y(e)\in path(v, m_Y(e))$ and $r_X(e)\in path(m_X(e), par(v))$.

**Proof:** Moving a vertex $r$ on $path(v, m_Y(e))$ away from the path to a descendant $r'$ increases its distance to $v$ and increases the distance-sum. So $r_Y(e)\in path(v, m_Y(e))$. Similarly, one can show that $r_X(e)\in path(m_X(e), par(v))$.

For an edge $e$, let $y_e$ denote the vertex farthest from $m_X(e)$ on $path(v, m_Y(e))$ with $d(m_X(e), y_e)\leq l_d$. Note that for any edge $e=(v, par(v))$ and vertex $y\in Y(e)$, $path(par(v), m_X(e)) \subseteq path(y, m)\cup J_T(e)$. So for a vertex $y\neq m$, let $x_y$ denote the vertex farthest from $y$ on $path(y, m)\cup J_T(e)$ with $d(y, x_y)\leq l_d$. Instead of finding all those $(r_X(e), r_Y(e))$, we produce the following two sets of pairs:

$$S_1=\{(m_X(e), y_e) \mid e\in E(T)\} \text{ and } S_2=\{(x_y, y) \mid y\in V(T)\}.$$

The following lemma shows that they are good enough for our purpose.

**Lemma 4.2** For every $(r_X(e), r_Y(e))$, there is a pair $(x, y)\in S_1\cup S_2$ such that $Sum_T(x, y)\leq Sum_{X(e)}(r_X(e))+Sum_{Y(e)}(r_Y(e))$.

**Proof:** Let $e=(v, par(v))$. If $d(m_X(e), r_Y(e))\leq l_d$, $y_e$ is the nearest vertex to $m_Y(e)$ on $path(v, m_Y(e))$, so $Sum_T(m_X(e), y_e) \leq Sum_{X(e)}(m_X(e)) + Sum_{Y(e)}(y_e) \leq Sum_{X(e)}(r_X(e)) + Sum_{Y(e)}(r_Y(e))$. Otherwise, for $y=r_Y(e)$, $x_y$ is the nearest vertex to $m_X(e)$ on $path(m_X(e), par(v))$, so $Sum_T(x_y, y) \leq Sum_{X(e)}(x_y) + Sum_{Y(e)}(y) \leq Sum_{X(e)}(r_X(e)) + Sum_{Y(e)}(r_Y(e))$.

$S_1\cup S_2$ has at most $2n$ pairs and finding them can be reduced to the tree marker problem after all $m_X(e)$ are computed. For each $(x_y, y)\in S_2$, we also need to find $e=BS(x_y, y)$ in order to compute $Sum_T(x_y, y)=Sum_{X(e)}(x_y)+Sum_{Y(e)}(y)$, according to Property 2.1 and 2.2. Finding all such bisectors can again be reduced to the tree marker problem. So we have the following, assuming that getting all $m_X(e)$ takes $t_x$ time and sorting $n$ edge lengths takes $t_e$ time.

**Theorem 4.1** The second generalization can be solved in $O(n+t_x+t_e) = O(n\log n)$ time.

## 5. Linear Time Algorithms

Now we study cases when linear-time algorithms exist for the 2-median problem and the two generalizations. A common obstacle is to find all $m_X(e)$, which has the following lower bound, proved in Appendix B.

**Theorem 5.1** Without assumption on vertex weights, finding all those $m_X(e)$ requires

$\Omega(n\log n)$ time in the comparison model, even when every edge has a unit length.

Auletta *et al.*'s 2-median algorithm in [1] computes all $m_X(e)$ without any assumption on vertex weights. Therefore, as mentioned in the introduction, their linear-time assertion on the running time is incorrect.

Theorem 5.1 suggests two possibilities for a more efficient algorithm. The first is to avoid finding all $m_X(e)$'s, and we consider this assuming that edge lengths are at most $k=n^{O(1)}$. Suppose that we have obtained every $m_Y(e)$. According to Property 2.1, if $(m_X(e), m_Y(e))$ is a 2-median, then $e=BS(m_X(e), m_Y(e))$. So to find $m_X(e)$, only the interval of vertices $v$ on $J_T(e)$ satisfying $e=BS(v, m_Y(e))$ needs to be considered. Sometimes we may fail to find $m_X(e)$ there, but it doesn't matter as such $(m_X(e), m_Y(e))$ is not a 2-median. Let $path(start(e), end(e))$ denote this interval with $start(e)$ being the one closer to $e$. After knowing the distance of every vertex to the root, finding all $(start(e), end(e))$'s takes $O(n)$ time by an integer sorting as $k=n^{O(1)}$ is assumed. One can check that each interval has length at most $2d(e)\leq 2k$ and contains at most $O(k)$ vertices, so $m_X(e)$ can be found in $O(\log k)$ time using a binary search. So the 2-median problem takes $O(n\log k)$ time. A similar idea works for the generalized problem in Section 3 too. The bottleneck there is to find $m_X(e)$ and then $r_X(e)$ for those $e\notin F$. Now we don't need $m_X(e)$. From Property 2.1, if $(r_X(e), r_Y(e))$ is a solution, then $e=BS(r_X(e), r_Y(e))$, so we search $r_X(e)$ from the interval on $path(m, c(T))\cup J_T(e)$ satisfying this condition. Thus, we have the following.

**Theorem 5.2** Both the 2-median problem and the generalization in Section 3 can be solved in $O(n\log k)$ time when edge lengths are at most $k=n^{O(1)}$. In particular, it takes linear time for constant $k$.

The second possibility is having some restriction on vertex weights, and we consider the case with vertex weights being polynomially bounded integers. Again, we root $T$ at $m(T)$. For each $e=(v, par(v))$, $m_X(e)$ is the vertex $u\in X(e)$ with the smallest $\delta_{X(e)}(u)=\delta_T(u)+w(T_v)>0$, or equivalently the smallest $\delta_T(u)>-w(T_v)$, according to Lemma 2.1. For those $e\notin T_\alpha$, $m_X(e)\in P(T_\alpha)$, and we can find all such $m_X(e)$ by sorting the set $\{-w(T_v) \mid v\notin T_\alpha\} \cup \{\delta_T(u) \mid u\in P(T_\alpha)\}$. For those $e\in T_\alpha$, $m_X(e)\in P(T_\beta)$, and we can find all such $m_X(e)$ by sorting the set $\{-w(T_v) \mid v\in T_\alpha\} \cup \{\delta_T(u) \mid u\in P(T_\beta)\}$. All those $\delta_T(u)$ and $w(T_v)$ can be computed in linear time via a bottom-up way. So we have the following.

**Theorem 5.3** Both the 2-median problem and the generalization in Section 3 can be solved in linear time when vertex weights are polynomially bounded integers.

Section 4's algorithm also sorts edge lengths, so we only have the following.

**Theorem 5.4** The generalized problem in Section 4 can be solved in linear time when all edge lengths and vertex weights are polynomially bounded integers.

## 6. Relaxation from Vertices to Points

In this section, we consider the relaxation by allowing 2-medians on points instead of just on vertices. We know that without constraints on 2-medians, this relaxation does not give smaller distance-sum. So we consider the two generalized problems discussed in Section 3 and 4.

All the properties in Section 3 still hold with respect to such relaxation, so the same algorithm also works here, and we have the following theorem.

**Theorem 6.1** The relaxed problem of the first generalization needs $O(n\log n)$ time.

Some work is needed for the second generalization. Our algorithm again is based on the link-deletion method, but now there is a new possibility that the pair of a 2-median lie on the same edge. So we first find two points $p_1(e)$ and $p_2(e)$ on each edge $e$ with $d(p_1(e), p_2(e))\leq l_d$ that minimize the distance-sum. After that, for each edge $e$, we want to find a point $r_X(e)$ on $X(e)$ and a point $r_Y(e)$ on $Y(e)$ with $d(r_X(e), r_Y(e))\leq l_d$ that achieve the smallest distance-sum. For the reason similar to that of Lemma 4.1, we know that for any $e=(v, par(v))$, $r_X(e)$ is on $path(m_X(e), par(v))$ and $r_Y(e)$ is on $path(v, m_Y(e))$.

We can assume $d(e)\leq l_d$, as $(r_X(e), r_Y(e))$ does not exist otherwise, and $d(m_X(e), m_Y(e))>l_d$, as $(r_X(e), r_Y(e))=(m_X(e), m_Y(e))$ otherwise. We can also assume that $d(r_X(e), r_Y(e))=l_d$ as otherwise moving $r_X(e)$ and $r_Y(e)$ apart further decreases the distance-sum. For convenience, let's say that $m_X(e)$, $par(v)$, $v$, and $m_Y(e)$ are lined up from left to right. Consider what happens when moving on $path(m_X(e), par(v))$ from a point $r_1$ to a point $r_1'$ with a small fixed distance $\Delta$. Moving $r_1$ left decreases $Sum_{X(e)}$ by a positive amount $dec(r_1)$, and $dec(r_1)$ gets smaller as $r_1$ comes closer to $m_X(e)$ because the partition of $X(e)$ becomes more balanced. Moving $r_1$ right increases $Sum_{X(e)}$ by a positive amount $inc(r_1)$, and $inc(r_1)$ gets larger as $r_1$ moves closer to $par(v)$. When moving a point $r_2$ on $path(v, m_Y(e))$, similar phenomena occurs with the difference that moving $r_2$ left now increases $Sum_{Y(e)}$. Note that $dec(r)\leq inc(r)$ for any point $r$.

We will do a binary search for $(r_X(e), r_Y(e))$ in iterations, each time updating two intervals $I_X$ and $I_Y$ for $r_X(e)$ and $r_Y(e)$ respectively, with $I_X= path(m_X(e), par(v))$ and $I_Y= path(v, m_Y(e))$ initially. After $I_X$ or $I_Y$ are reduced to one edge, $r_X(e)$ and $r_Y(e)$ can then be found easily. In each iteration, we select vertices $r_1$ and $r_2$ to cut $I_X$ and $I_Y$ each into two halves of almost equal number of vertices. Two cases to consider:

**(1) $dec(r_1)\leq inc(r_2)$.** Suppose $d(r_1, r_2)\geq l_d$. Let $r_2'$ denote the point on $path(v, m_Y(e))$ with $d(r_1, r_2')=l_d$, and note that $inc(r_2')\geq inc(r_2)$. Moving $(r_1, r_2')$ left for a distance $b$ increases the distance-sum by at least $(inc(r_2')-dec(r_1))b/\Delta\geq 0$, so we can ignore the half to the left of $r_1$ for $r_X(e)$, and update $I_X$ to be the remaining half. Otherwise suppose $d(r_1, r_2)<l_d$. Let $r_1'$ denote the point on $path(m_X(e), u)$ with $d(r_1', r_2)=l_d$, and

note that $dec(r_1')\leq dec(r_1)$. Moving $(r_1', r_2)$ left for a distance $b$ increases the distance-sum by at least $(inc(r_2)-dec(r_1'))b/\Delta\geq 0$, so we can ignore the half to left of $r_2$ for $r_Y(e)$, and update $I_Y$ to be the remaining half.

**(2)** $\boldsymbol{dec(r_1)>inc(r_2)}$**.** Note that $dec(r_2)\leq inc(r_2)<dec(r_1)\leq inc(r_1)$. Thus, this is just case 1 with $r_1$ and $r_2$ switched, and a similar action can be taken.

So we first find all $(p_1(e), p_2(e))$ in $O(n)$ time, and all $(m_X(e), m_Y(e))$ in $O(n\log n)$ time. Then we use binary searches as described above to find $r_X(e)$ and $r_Y(e)$ for every edge $e$, but with all the $n$-1 binary searches carried out in parallel. During each iteration, we find all $n$-1 middle vertices at once by reducing the task to the tree maker problem with unit edge length, which only takes $O(n)$ time. There are at most $O(\log n)$ iterations, so $O(n\log n)$ time suffices. Finally we choose the pair $(p_1(e), p_2(e))$ or $(r_X(e), r_Y(e))$ with the smallest distance-sum. So we have the following theorem.

**Theorem 6.2** The relaxed problem of the second generalization needs $O(n\log n)$ time.

## 7. Future Work

For the $p$-median problem on trees, Tamir [14] has an $O(pn^2)$-time algorithm. Can it be solved in $o(n^2)$ time for any $p>2$? We have an $\Omega(n\log n)$-time lower bound for the link-deletion method, but we are more interested in a lower bound for the 2-median problem and its generalizations. Can they be solved in $o(n\log n)$ time? Wang *et al.* [18] solved both generalizations in $O(\log n)$ parallel time but with $O(n^2)$ work. Can one design more efficient parallel algorithms for them? A simple polynomial solution to the second generation was proposed in the introduction for general $p$. Can one design more efficient algorithms?

## References


[1] V. Auletta, D. Parente, and G. Persiano, Dynamic and static algorithms for optimal placement of resources in a tree, *Theoretical Computer Science*, vol. 165, pp. 441-461, 1996.

[2] R. E. Burkard, E. Çela, and H. Dollani, 2-Medians in trees with pos/neg weights, *Discrete Applied Mathematics*, vol. 105, pp. 51-71, 2000.

[3] E. Erkut, R. L. Francis, and T. J. Lowe, Locating two facilities on a tree subject to distance constraints, *Transportation Science*, vol. 22, no. 3, pp. 199-208, 1988.

[4] H. N. Gabow and R. E. Tarjan, A linear-time algorithm for a special case of disjoint set union, *Journal of Computer and System Sciences*, vol. 30, pp. 209-

221, 1985.

[5] Z. Galil and G. F. Italino, Data structures and algorithms for disjoint set union problems, *ACM Computing Surveys*, vol. 23, no. 3, pp. 319-344, 1991.

[6] B. Gavish and S. Sridhar, Computing the 2-median on tree networks in $O(n\lg n)$ time, *Networks*, vol. 26, pp. 305-317, 1995.

[7] A. J. Goldman, Optimal center location in simple networks, *Transportation science*, vol. 5, pp. 212-221, 1971.

[8] S. K. Guta and A. P. Punnen, Group center and group median of a tree, *European Journal of Operational Research*, vol. 65, pp. 400-406, 1993.

[9] S. L. Hakimi, Optimal distribution of switching centers in communication networks and some related graph theoretical problems, *Operations Research*, vol. 13, pp. 462-475, 1965.

[10] G. Y. Handler and P. Mirchandani, *Location on Networks*, MIT Press, Cambridge, MA, 1979.

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

[12] P. B. Mirchandani and A. Oudjit, Localizing 2-medians on probabilistic and deterministic tree networks, *Networks*, vol. 10, pp. 329-350, 1980.

[13] B. C. Tansel, R. L. Francis, and T, J. Lowe, Location of networks: A survey, *Management Science*, vol. 29, pp. 482-511, 1983.

[14] A. Tamir, An $O(pn^2)$ algorithm for the $p$-median and related problems on tree graphs, *Operations Research Letters*, vol. 19, pp. 59-64, 1996.

[15] A. Tamir, D. Perez-Brito, and J. A. moreno-Perez, A polynomial algorithm for the $p$-centdian problem on a tree, *Networks*, vol. 32, pp. 255-262, 1998.

[16] B.-F. Wang, Efficient parallel algorithms for optimally locating a path and a tree of a specified length in a weighted tree network, *Journal of Algorithms*, vol. 34, pp. 90-108, 2000.

[17] B.-F. Wang, Finding a two-core of a tree in linear time, *SIAM Journal on Discrete Mathematics*, accepted.

[18] B.-F. Wang, S.-C. Ku, and K.-H. Shi, Cost-optimal parallel algorithms for the tree bisector problem and applications, *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 9, pp. 888-898, 2001.

## Appendix A. Proof of Theorem 2.1

Let's do some preprocessing first. Root the tree at an arbitrary vertex $r$, and find the least common ancestor for every vertex pair, which takes only linear time. Then, replace each 3-tuple $(x, l, y)$ by the pair $(v, t(v))$ which is $(x, d(x, r)-l)$ if $l \le d(x, LCA(x, y))$ and $(y, d(y, r)+l-d(x, y))$ otherwise, indicating that the point we want is above $v$ with distance $t(v)$ to $r$. For simplicity, we assume that no vertex $v$ appears in two pairs; otherwise, we can simply attach some copy nodes of $v$ to $par(v)$.

The target point for each pair $(v, t(v))$ can be found in $O(n+q\log n)$ time by simply performing a depth-first traversal on $T$. During the traversal, we maintain the path from root to the current vertex in an array such that the target point for each pair $(v, t(v))$ can be found by binary search.

Next, an $O(n+t_e)$-time algorithm is presented. We borrow the idea from *Wang et al.*'s algorithm for the tree bisector problem [18], which in fact is a special case of the tree marker problem. Consider the Euler tour $U$ of $T$, which is obtained by performing a DFS on $T$, recording the vertices visited. Let $U[i]$ denote the *i-th* vertex in the tour. For a pair $(v, t(v))$, we want to find $v$'s highest ancestor $u$ with $d(u, r)>t(v)$. Let $w=par(u)$. Note that $d(w, r)\le t(v)$ and $d(u', r)>t(v)$ for every vertex $u'$ in the subtree $T_u$. Also note that in $U$, there is an appearance of $w$ right before the Euler tour of the subtree $T_u$. If we go backward from the first appearance of $v$ in $U$, $w$ is first vertex we encounter with $d(w, r)\le t(v)$. Let $F$ and $X$ be the arrays with $F[i]=t(U[i])$ and $X[i]=d(r, U[i])$. Now the remaining task can be easily reduced to the following *f-left-match problem*, introduced by Wang *et al.* [18]. The input is two arrays $X$ and $F$, each of $n$ numbers. The output is an array $M$ such that $M[k]$ is the largest index $l<k$ with $X[l]\le F[k]$ and $M[k]=\phi$ if no such index exists. $X[M[k]]$ is called $F[k]$'s *target element*.

Given input $X$ and $F$, we first stable-sort their concatenation increasingly into a new array $W$. Let $Z$ be the array of $2n$ elements with $Z[i]=1$ if $W[i]$ is from $X$ and $Z[i]=0$ otherwise. Let $D$ be the array of $2n$ elements with $D[i]$ being the index of $W[i]$ in its original array, $X$ or $F$. An example is shown in Figure A.1. Using some data structure $S$, we do the following for index $i$ from 0 to $2n$-1. If $Z[i]=1$, insert $D[i]$ into $S$. Otherwise, search the largest number $l<D[i]$ in $S$ and set $M[D[i]]=l$. This assignment is correct because at this point, $S$ contains exactly those indices $l$ such that $X[l]\le F[D[i]]$.

The complexity depends on how $S$ is implemented. This is related to the *interval split-find problem* introduced in [5], as the set of inserted numbers is in fact $I_n=\{0, 1, ..., n-1\}$. An *interval* of $I_n$ is a set of consecutive numbers in $I_n$. The problem is to maintain a data structure that represents some partition of $I_n$ and supports two types of operations: *find*$(x)$, which returns the interval containing $x$, and *split*$(x)$, which splits the interval of $x$ into two, one for those less than $x$ and one for the rest. Gabow and Tarjan [4] had an $O(n)$-time algorithm for $O(n)$ operations, which we will build our data structure $S$ upon. Initially we have only one interval $I_n$. The idea is that if we insert each $D[j]$ in $S$ by calling *split*$(D[j])$, then given $D[i]$, the largest $D[j]<D[i]$ in $S$ is just the smallest number in the interval containing $D[i]$. The only thing we need to take care is to record the

smallest number of each interval, using an array $SS$ updated in the following way. Whenever a $D[j]$ is to be inserted, let $s$=$SS$[$find$($D[j]$)], call $split$($D[j]$), set $SS$[$find$($D[j]$-1)]=$s$, and set $SS$[$find$($D[j]$)]=$D[j]$. There are $O(n)$ operations and only linear time is needed. Thus, we have the following lemma, assuming $t_a$ is the time for stable-sorting the concatenation of $X$ and $F$, which immediately implies Theorem 2.1.

**Lemma A.1** The $f$-left-match problem can be solved in $O(n+t_a)$ time.

## Appendix B. Proof of Theorem 5.1

Consider the *multi-search problem* defined by Wang *et al.* [18]. The input is an array $P$ of $p$ unsorted numbers and an array $Q$ of $q$ distinct sorted numbers. The output is an array $R$ of $p$ integers, with $R[i]$, for $1\leq i\leq p$, recording the number of elements in $Q$ smaller than $P[i]$. Wang *et al.* proved an $\Omega(p\log q)$-time lower bound in the comparison model for this problem. We will reduce this problem in linear time to the link-deletion method.

Assume that $p\geq 2$. Let $s=\Sigma_{1\leq i\leq p}P[i]$, and we can assume w.l.o.g. that $s>Q[q]$, as otherwise an entry of a large value can be added to $P$. Construct a tree $T$ of $p+q+2$ vertices with unit edge lengths in the following way. $T$ has a root $z$ with $w(z)=s$. For $1\leq i\leq p$, there is an edge $e_i$ connecting $z$ to a leaf $v_i$ with $w(v_i)=s-P[i]$. There is a path from a leaf $u_0$ to $z$, passing through vertices $u_1$, $u_2$, …, $u_q$. Let $w(u_0)=(p-1)s+Q[q]/2$, $w(u_1)=Q[1]/2$, and $w(u_i)=(Q[i]-Q[i-1])/2$ for $2\leq i\leq q$. See Figure B.1 for an illustration.

Let $H$ denote the subtree of $z$ rooted at $u_q$. Note that $w(T\backslash H) = w(z)+\Sigma_{1\leq i\leq p}w(v_i) = s+(p-1)s = ps$, and $w(H) = \Sigma_{0\leq i\leq q}w(u_i) = (p-1)s+Q[q] < ps$; so the median of $T$ must lie on $T\backslash H$. As each $v_i$ has negative $\delta_T(v_i)$, $z$ is the median of $T$. $H$ is the heaviest subtree of $T$, so for $1\leq i\leq p$, $m_X(e_i)$ must lie on $H$ if edge $e_i$ is deleted. Now $w(X(e_i)\backslash H) = w(z)-w(v_i)+\Sigma_{1\leq j\leq p}w(v_j) = s-(s-P[i])+(p-1)s = (p-1)s+P[i]$, for $1\leq i\leq p$, and $\delta_H(u_k) = \Sigma_{0\leq j\leq k}w(u_j)-\Sigma_{k<j\leq q}w(u_j) = ((p-1)s+Q[q]/2+Q[k]/2)-(Q[q]/2-Q[k]/2) = (p-1)s+Q[k]$, for $1\leq k\leq q$. As $m_X(e_i)$ is the vertex $u_k$ on $H$ with the smallest nonnegative $\delta_H(u_k)-w(X(e_i)\backslash H)=Q[k]-P[i]$, it gives the smallest $k$ with $Q[k]\geq P[i]$. So finding $m_X(e_i)$ for every edge $e_i$ solves the multi-search problem on the input $P$ and $Q$.

Given any two arrays $P$ and $Q$, we can construct the corresponding tree $T$ in linear time. As the multi-search problem with $p=q=n/2$ has an $\Omega(n\log n)$-time lower bound, so does the link deletion method.

## Appendix C. The Group Median Problem

Given a tree $T=(V, E)$ and a vertex partition $\{S_1, S_2, ..., S_k\}$ of $V$, a point $g\in V$ is a *group median* of $T$ if it minimizes $\Sigma_{1\leq i\leq k}$ $\{\max\{d(v, g)|\ v\in S_i\}\}$, over all vertices of $T$. Gupta and Punnen [8] presented two algorithms for finding a group median of a tree, running in $O(n\log n)$ time and $O(nk)$ time, respectively. Let $X_i$, $1\leq i\leq k$, be the smallest subtree of $T$ covering all vertices in $S_i$. The most critical step of their second algorithm is to find a center $c(X_i)$ for every $X_i$. All the other steps require only linear time. They implemented the finding of all $c(X_i)$ in $O(kn)$ time. In the following, we give a more efficient implementation for the finding. We preprocess $T$ such that $d(u, v)$ can be determined in $O(1)$ time for any given $u$, $v\in V$. For each $X_i$ we find a diameter path $dm(X_i)$ as follows. Select an arbitrary leaf $a$ of $X_i$ and find the leaf $b$ farthest from $a$ in $X_i$. Then, find the leaf $c$ farthest from $b$ in $X_i$. The path from $b$ to $c$ is a diameter path [10]. All leaves of $X_i$ are vertices of $S_i$. Thus, each $X_i$ has at most $|S_i|$ leaves. Therefore, the finding of all $dm(X_i)$ requires $O(n)$ time. For each $X_i$, the midpoint of $dm(X_i)$ is $c(X_i)$ [10]. Thus, all $c(X_i)$ can be found by applying our tree marker algorithm.

**Theorem C.1** The group median problem can be solved in $O(n+\min\{k\log n, t_e\})=O(n\log n)$ time. In particular, it takes linear time when $k=O(n/\log n)$ or when all edge lengths are polynomially bounded integers.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| *X* | 1 | 3 | 8 | 5 | 7 | | | | |
| *F* | * | 2 | 4 | 2 | 6 | | | | |
| *M* | * | 0 | 1 | 0 | 3 | | | | |
| *W* | 1 | 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *Z* | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| *D* | 0 | 1 | 3 | 1 | 2 | 3 | 4 | 4 | 2 |

*: Don't care

**Figure A.1** An example of the arrays *M*, *W*, *Z*, and *D* for the arrays *X* and *F*.

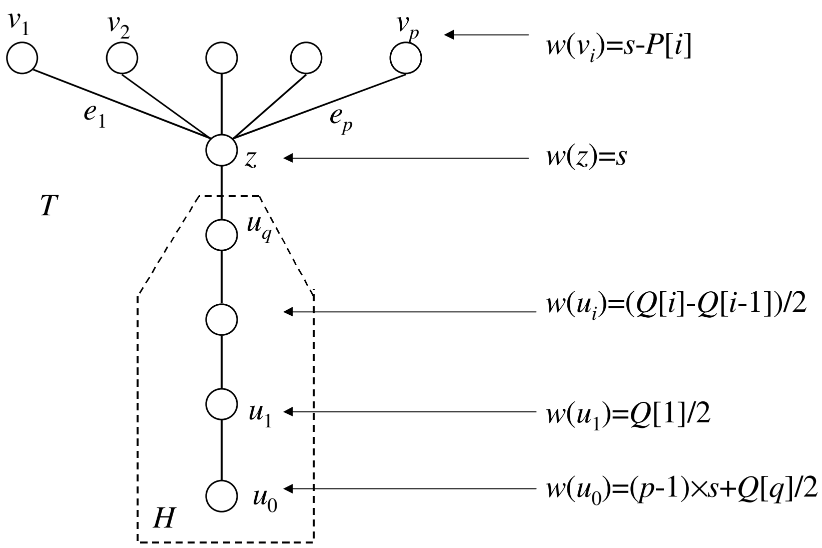


**Figure B.1** A transformation from the multi-search problem to the 2-median problem.