Efficient Algorithms for Finding 2-Medians of a Tree

Aissa Oudjit and Matthias F. Stallmann[⋆]

North Carolina State University

Abstract. The p-median problem on general networks has been studied since the 1960s. Kariv and Hakimi [10] showed that this problem is NP-hard even if the network is a planar graph of maximum degree 3. In the case of tree networks the p-median problem is solvable in polynomial time. Kariv and Hakimi [10] developed an algorithm that computes a solution in $\mathrm{O}(p^2 n^2)$ time. The running time was improved to $\mathrm{O}(pn^2)$ by Tamir [16] and later to $\mathrm{O}(n \lg^{p+2} n)$ by Benkoczi and Bhattacharya [2].

Better bounds are known for the special cases (on trees) where $p = 1$ or 2. Goldman [6] gave an $\mathrm{O}(n)$ algorithm for the 1-median problem on trees. The 2-median problem was studied by Mirchandani and Oudjit [11], whose localization properties were later used to improve the $\mathrm{O}(n^2)$ bound (derived from the general tree case) to $\mathrm{O}(n \lg n)$ – see papers by Hämäläinen [9] and Gavish and Sridhar [5].

We present a framework for solving the 2-median problem on trees, building on earlier work. Our framework leads to several algorithms with $o(n \lg n)$ runtime, i.e., better than the current best-known $\mathrm{O}(n \lg n)$ runtime, in common special cases. The time bounds are:

(i) $\mathrm{O}(n \lg w_{\max} / \lg n)$, where $w_{\max}$ is the largest largest node weight, which is linear when node weights are bounded by a polynomial in n;

(ii) $\mathrm{O}(n \lg n_L)$, where n_L is the number of leaves in in the tree;

(iii) $\mathrm{O}(n d_{\max})$, where $d_{\max}$ is the maximum edge length, which is linear when edge lengths are bounded by a constant;

and (iv) $\mathrm{O}(n \lg \ell)$, where ℓ is the number of nodes on the *trunk*, an easily identified path that is guaranteed to contain at least one of the two medians.

1 Introduction

Since its formulation by Hakimi [7,8], the p-median problem on networks continues to be an area of active research. The problem is to find the locations of p facilities on a network so as to minimize the transportation costs from each demand point (node) to its nearest facility. Hakimi [8] showed that the optimal facility locations for a general network exist at the nodes of the network, leading to the following combinatorial definition. Given a network $G = (V, E)$, where each node $i \in V$ has a non-negative weight (demand) w_i and each (undirected) edge $ij \in E$ has a positive length (distance) d_{ij}, find $m_1, \ldots, m_p$ such that the total weighted distance

$$\sum_{i \in V} \left(\min_{1 \le k \le p} \mathrm{d}[i, m_k] \right) \cdot w_i$$

is minimized, where $\mathrm{d}[i, m_k]$ is the length of a shortest path from i to m_k, the sum of the edge distances along that path. The m_k are not necessarily unique; where they are not, we can choose them arbitrarily without affecting the validity of any theoretical results to follow.

[⋆] mfms@ncsu.edu

The combinatorial nature of the p-median problem led some researchers to formulate it as an integer linear program [14]. While such formulation made the problem more manageable, standard solution techniques such as branch and bound and dynamic programming (see [13]) lead to prohibitive runtimes and various heuristics (see [12]) are unsatisfactory because solution quality suffers. Significant improvements for general networks are unlikely since the problem is NP-hard, even for degree-3 planar graphs with unit edge lengths [10]. Consequently, much of the research on the p-median problem in the past decades has focused on special classes of graphs, most notably trees [2,4,10,16].

Goldman [6] derived a linear time algorithm for finding the 1-median of a tree with arbitrary edge lengths and node weights. Using Goldman's algorithm, Dearing, Frances, and Low [4] gave efficient algorithms for solving, in tree networks, a variety of problems having convexity properties. Kariv and Hakimi [10] gave an $\mathrm{O}(p^2n^2)$ algorithm for the p-median problem on general trees. This was later improved to $\mathrm{O}(pn^2)$ by Tamir [16] and $\mathrm{O}(n \lg^{p+2} n)$ by Benkoczi and Bhattacharya [2]. The latter, which uses a *spine decomposition* of the tree, is an improvement over the prior quadratic time bounds if p is fixed.

Mirchandani and Oudjit [11] were the first to study the 2-median problem on trees in detail. They derived several *localization properties* for the positions of the 2-medians relative to the 1-median and proposed an $\mathrm{O}(n^2)$ algorithm based on *improved link deletion*. While this time bound is not an improvement over the previously known quadratic one, the theory leading to it is the basis for later $\mathrm{O}(n \lg n)$ algorithms by Hämäläinen [9], by Gavish and Sridhar [5], and for the sub-$n \lg n$ special cases reported here. An independent, more general, $\mathrm{O}(n \lg n)$ algorithm was reported by Breton [3].

The remainder of the paper is organized as follows.

- Section 2 presents some notation for the p-median problem on trees and lays the groundwork for our 2-median algorithms.
- Section 3 presents results relevant to our approach to the 2-median problem with useful definitions and notation.
- Section 4 gives an algorithm for computing one of the two medians in linear time.
- Sections 5–7 give three algorithms that yield $o(n \lg n)$ – better than $\mathrm{O}(n \lg n)$ – time bounds in common special cases.
- Section 8 summarizes our results.
- The appendix illustrates our algorithms on three examples.

2 Preliminaries

The input to the p-median problem consists of a tree T and (i) for each node i of T, a non-negative *weight* w_i; (ii) for each edge ij of T, a positive *distance* (or length) d_{ij}. We define $\mathrm{w}(S)$, where S is any subtree[1] of T to be the *total weight* of nodes in S, the sum of all w_i for $i \in S$; and $\mathrm{d}[x, y]$ to be the *total distance* between x and y, i.e., the sum of d_{jk} for all edges jk on the *unique* path between x and y. For any subtree S, let $\mathrm{cost}(S, x)$ be the total *cost*, i.e., weighted distance, of S with respect to x:

$$\sum_{i \in S} \mathrm{d}[i, x] w_i$$

[1] Throughout, a subtree is treated as a set of *nodes*; induced edges are implicitly included.

So when $S = T$, a 1-median m of T is a node that minimizes $\mathrm{cost}(T, m)$.

Some general notation. Each node j in T has $\deg(j)$ = the *degree* of j, the number of edges incident on j. A node j is a *leaf* if $\deg(j) = 1$. Knowing that, for any two nodes i and j, there is a unique path between them, we use interval notation for paths: $[i, j]$ to include both i and j; half-open intervals $(i, j]$ and $[i, j)$ if i, respectively j is not included; and (i, j) if neither endpoint is included. The (half-)open intervals represent empty paths if $i = j$.

Suppose $m_1, \ldots, m_p$ are p medians of T. and let $S_1, \ldots, S_p$ partition the nodes of T so that node $x \in S_i$ has $\mathrm{d}[x, m_i] < \mathrm{d}[x, m_j]$ for all $j \neq i$, and, if $\mathrm{d}[x, m_i] = \mathrm{d}[x, m_j]$, let x be in S_i, where $i < j$, i.e., put x into the set with smaller index if there is a tie. If node x is in S_i we say that m_i *serves* x or that x is served by m_i.

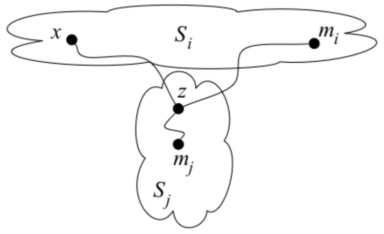


Fig. 1. The set of nodes closest to a given median m_i is not a forest.

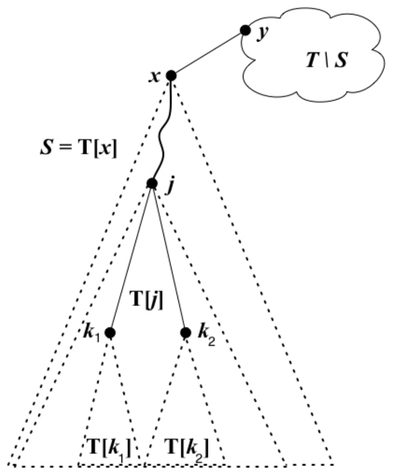
Lemma 1. *If S_i is the set of nodes served by m_i then S_i is a sub*-tree *of T.*

Proof. Suppose that S_i is a forest instead of a tree, and that node x is in a tree of S_i different from the one containing m_i. Path $[m_i, x]$ in T therefore includes node $z \in S_j \neq S_i$ and either $\mathrm{d}[z, m_j] < \mathrm{d}[z, m_i]$ or $\mathrm{d}[z, m_j] = \mathrm{d}[z, m_i]$ and $j < i$. Now $\mathrm{d}[x, m_i] = \mathrm{d}[x, z] + \mathrm{d}[z, m_i]$ and $\mathrm{d}[x, m_i] = \mathrm{d}[x, z] + \mathrm{d}[z, m_i]$. So either $\mathrm{d}[x, m_j] < \mathrm{d}[x, m_i]$ or $\mathrm{d}[x, m_j] = \mathrm{d}[x, m_i]$ and $j < i$. This implies that $x \in S_j$ rather than in S_i, a contradiction. See Fig. 1. $\square$

For the special case of the 2-median problem we have the following. Define the *midpoint* x of path $[i, j]$ to be such that $\mathrm{d}[i, x] = \mathrm{d}[x, j]$. In general x is not a node – it may lie along an edge.

Lemma 2. *The midpoint on the path $[m_1, m_2]$, where m_1 and m_2 are 2-medians of T, lies either along the edge that separates S_1 from S_2 or on one of its endpoints.*

Proof. Let $x_1 x_2$ with $x_i \in S_i$ be the edge that separates S_1 from S_2 and suppose that the midpoint between m_1 and m_2 is along some other edge yz. Without loss of generality both y and z are in S_2 and neither y nor z is one of the x_i's. Furthermore assume, again without loss of generality, that $\mathrm{d}[y, m_1] \leq \mathrm{d}[z, m_1]$. Then either y is closer to m_1 than it is to m_2 or it is equidistant from them. In either case, it should be in S_1, a contradiction. $\square$



The cut edge xy separates S from $T \setminus S$. The wavy line shows the path $[x, j]$ from x to node $j \in S$; since $k_1, k_2 \notin [x, j]$ they are children of j and their subtrees, $\mathrm{T}[k_1]$ and $\mathrm{T}[k_2]$ are subtrees of $\mathrm{T}[j]$ induced by removal of edges jk_1 and jk_2, respectively.

Fig. 2. A subtree S, induced by edge xy.

3 Theoretical Results

Let xy be any edge of T and let S and $T \setminus S$ be the subtrees induced when *cut edge* xy is removed. Suppose $x \in S$. In the context of the 2-median problem it is useful to treat x as the root of subtree S and define the following for any node $j \in S$.

- k is a *child* of j if k is adjacent to j but not on the path $[x, j]$;
- the subtree rooted at k, where k is a child of j, denoted $\mathrm{T}[k]$, is the subtree formed when cut edge jk is removed; in the special case where k is the root of S, $\mathrm{T}[k] = S$.

Fig. 2 illustrates these concepts.

3.1 Basic concepts

We begin with a fundamental lemma by Goldman [6]; both lemma and proof are restated here.

Lemma 3. *If* $\mathrm{w}(S) \geq \mathrm{w}(T \setminus S)$ *then at least one 1-median of* T *is in* S*. Conversely, if* m *is a 1-median of* T *and* $\mathrm{w}(S) \geq \mathrm{w}(T \setminus S)$ *then* m *must be in* S*.*

Proof. Let z be *any* node in $T \setminus S$. For the first part it suffices to prove $\text{cost}(T, z) \geq \text{cost}(T, x)$, where xy is the edge separating S from $T \setminus S$ and $x \in S$.

$$\begin{aligned}
\text{cost}(T, z) &= \text{cost}(S, z) + \text{cost}(T \setminus S, z) \\
&= [\text{w}(S) \cdot \text{d}[z, x] + \text{cost}(S, x)] + \text{cost}(T \setminus S, z) && \text{difference between having median} \\
& && \text{of } S \text{ at } z \text{ versus } x \\
&\geq \text{w}(T \setminus S) \cdot \text{d}[z, x] + \text{cost}(S, x) + \text{cost}(T \setminus S, z) && \text{by assumption} \qquad \textbf{(*)} \\
&= \text{cost}(S, x) + [\text{w}(T \setminus S)\text{d}[z, x] + \text{cost}(T \setminus S, z)] && \text{regrouping} \\
&= \text{cost}(S, x) + \textstyle\sum_{j \in T \setminus S} w_j \cdot [\text{d}[x, z] + \text{d}[j, z]] && \text{definition of cost} \\
&\geq \text{cost}(S, x) + \textstyle\sum_{j \in T \setminus S} w_j \cdot \text{d}[x, j] && \text{triangle inequality (equality holds} \\
& && \text{if no backtracking on the path)} \\
&= \text{cost}(S, x) + \text{cost}(T \setminus S, x) && \text{by definition of cost} \\
&= \text{cost}(T, x)
\end{aligned}$$

For the second, the contrapositive is: if $\text{w}(S) < \text{w}(T \setminus S)$ then S does not contain a 1-median of T. This follows by symmetry: switch the roles of S and $T \setminus S$ and change the $\geq$ in the line marked **(*)** to $>$. □

Corollary 1. *T has two 1-medians if and only if there exists $S \subset T$ with $\text{w}(S) = \text{w}(T)/2$. And, unless there is a path of degree-2 weight-0 nodes between two subtrees, T cannot have more than two 1-medians.*[2]

Corollary 2. *If j is not a 1-median of $\text{T}[j]$, then there is a 1-median of j in $\text{T}[k]$ for some k with $\text{w}(\text{T}[k]) \geq \text{w}(\text{T}[j])/2$.*

Corollary 2 motivates the following definition.

Definition 1. *The* extended trunk *of subtree $\text{T}[j]$ is denoted by $\text{et}(j)$ and defined recursively as follows:*
(i) $\text{et}(j) = j$ if $\text{T}[j] = \{j\}$, i.e., j is a leaf
(ii) $\text{et}(j) = j$ followed by $\text{et}(k^*)$, where $\text{w}(\text{T}[k^*]) \geq \text{w}(\text{T}[k])$ for any child k of j.

Ties can be broken arbitrarily in (ii). Put more directly, the extended trunk is derived by following a path from j and repeatedly moving to a largest weight subtree until we reach a leaf. Corollary 2 tells us that a the 1-median of any subtree has to be on its extended trunk. This is a much weaker statement than the corollary – it says nothing about the weight having to be at least half, but it helps us localize the median of a subtree in the algorithms.

3.2 Locating each median

After making appropriate arbitrary choices to define $\text{et}(j)$, we can specifically pinpoint a canonical 1-median of $\text{T}[j]$.

Definition 2. *The canonical 1-median of $\text{T}[j]$, denoted $\text{m}(j)$, is the last node k on $\text{et}(j)$ (node farthest from j) with $\text{w}(\text{T}[k]) \geq \text{w}(\text{T}[j])/2$.*

[2] Such a path can be collapsed into a single edge without changing the nature of the problem.

Lemma 4. *The canonical 1-median of* $\mathrm{T}[j]$ *is on the path* $[j, \mathrm{m}(k^*)]$*, where* k^* *is the child of* j *on* $\mathrm{et}(j)$*.*

Proof. Suppose, for the sake of contradiction, that $\mathrm{m}(j) = k$ for some $k \notin [j, \mathrm{m}(k^*)]$, i.e., k is farther from j than $\mathrm{m}(k^*)$ on $\mathrm{et}(j)$. This means, by Definition 2, that $\mathrm{w}(\mathrm{T}[k]) < \mathrm{w}(\mathrm{T}[j])/2$. But Lemma 2 implies that any subtree containing a 1-median of $\mathrm{T}[j]$ must have weight at least half that of $\mathrm{T}[j]$; so we have a contradiction. □

Lemma 5. *If* $\mathrm{T}[j]$ *contains* m*, a 1-median of* T*, then* $\mathrm{m}(j)$ *is on the path* $[m, \mathrm{m}(\mathrm{T}[m^*])]$*, where* m^* *is the child of* m *on* $\mathrm{et}(j)$*.*

Proof. The fact that $\mathrm{m}(j)$ cannot come after $\mathrm{m}(m^*)$ follows a fortiori from Lemma 4. Lemma 3 implies that $\mathrm{w}(\mathrm{T}[m]) \geq \mathrm{w}(T)/2$. Since $\mathrm{T}[j]$ is a strict subset of T, we know $\mathrm{w}(\mathrm{T}[m]) \geq \mathrm{w}(\mathrm{T}[j])/2$. So, applying Corollary 2, $\mathrm{m}(j)$ must be in $\mathrm{T}[m]$, i.e., not before m on the extended trunk. □

The following helps us further restrict the location of the median in the subtree containing m. It is used by one of our algorithms.

Lemma 6. *Let* S *and* S' *be two subtrees, both containing* m*, and suppose* $\mathrm{w}(S) \leq \mathrm{w}(S')$*. Then* $\mathrm{m}(S')$ *is on the path* $[m, \mathrm{m}(S)]$*.*

Proof. We know $\mathrm{m}(S)$ is on path $P = [m, \mathrm{m}(m^*)]$, and, more precisely, it is the last node on P with $\mathrm{w}(\mathrm{T}[\mathrm{m}(S)]) \geq \mathrm{w}(S)/2$. Any predecessor k of $\mathrm{m}(S)$ on P, including $\mathrm{m}(S)$ itself, has $\mathrm{w}(\mathrm{T}[k]) \geq \mathrm{w}(S)/2 \geq \mathrm{w}(S')/2$. So the last node k' on P with $\mathrm{w}(\mathrm{T}[k']) \geq \mathrm{w}(S')/2$ must be a successor of $\mathrm{m}(S)$. □

Note that in this case S does not have to be a subtree of S'.

We use the following definitions as a basis for locating medians in the two subtrees of interest, one containing m, the 1-median of T, the other not.

Definition 3. *Let* $v_1, \ldots, v_d$ *be the nodes adjacent to* m *and let* $T_1, \ldots, T_d$ *be the subtrees resulting from removal of edges* $mv_1, \ldots, mv_d$*, respectively. Choose* T_1 *and* T_2 *so that* $\mathrm{w}(T_1) \geq \mathrm{w}(T_2) \geq \mathrm{w}(T_i)$ *for* $i > 2$*. The* trunk *of* T *is the path* $[\mathrm{m}(T_1), \mathrm{m}(T_2)]$ *passing through* m*, or stopping at* m *if* T_2 *does not exist.*

The following is a direct consequence of Lemma 5.

Theorem 1. *Suppose* m_1, m_2 *are 2-medians of* T*. Then one of the following holds: (i)* $m_1 \in T_1$ *and* $m_2 \in [m, \mathrm{m}(T_2)]$*; or (ii)* $m_1 \in T \setminus T_1 \setminus m$ *and* $m_2 \in [m, \mathrm{m}(T_1)]$*.*

Proof. In case (i) T_2 plays the role of $\mathrm{T}[m^*]$ in Lemma 5: $\mathrm{T}[j] = T \setminus T_1$ includes m and T_2 and, with T_1 not in contention, $\mathrm{et}(m)$ descends into T_2. In case (ii) $\mathrm{T}[j]$ is $T_1 \cup m$ and $\mathrm{et}(m)$ descends into T_1. □

So one of the 2-medians is known to be on the trunk, specifically the one that serves m.

Definition 4. *If* $\mathrm{T}[j]$ *does not contain the median* m *of* T *we refer to* $\mathrm{m}(j)$ *as the* near median *of* j*. The near median serves* j *but not* m *and is referred to as* m_1 *in Theorem 1.*

Definition 5. *If* $\mathrm{T}[k]$ does *contain the median* m *of* T *and* j *is the node adjacent to* k *that is* not *a child of* k *in* $\mathrm{T}[k]$*, then* $\mathrm{m}(k)$ *is the* far median *of* j*. The far median serves* m *and* k *but not* j *and is referred to as* m_2 *in Theorem 1. Henceforth, we use the notation* $\mathrm{m}'(j)$ *for the far median.*

Lemma 5 tells us that, if j is in T_1 then the far median of j, i.e., $\mathrm{m}(T \setminus \mathrm{T}[j])$, is on the segment of the trunk between m and $\mathrm{m}(T_2)$. Conversely, if $j \notin T_1$, then the far median of j is between m and $\mathrm{m}(T_1)$.

The algorithms are more easily described if we direct all edges toward m, from child to parent.

Definition 6. *If edge* jk *separates* T *into* S *and* $T \setminus S$*, where* $j \in S$ *and* $k, m \in T \setminus S$*, then we refer to* k *as the* parent *of* j*, denoted* p_j*.*

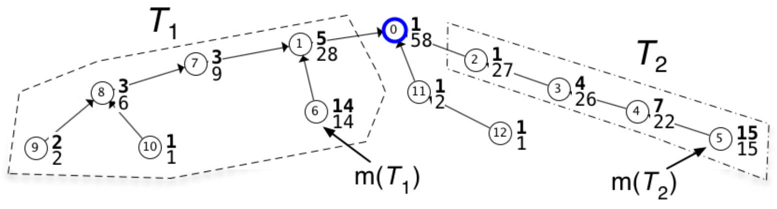


Fig. 3. A tree T after the 1-median has been identified. Node numbers are inside the circles – node 0 is the 1-median.

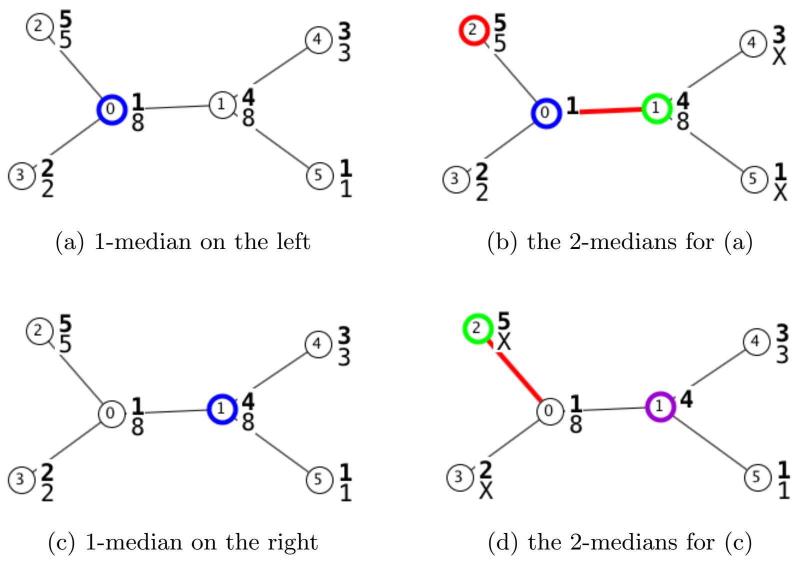
The concepts just described are illustrated in Fig. 3. The top numbers outside the circles are the node weights, the bottom numbers are the subtree weights $\mathrm{w}(\mathrm{T}[j])$ for each $j \neq m$ and $\mathrm{w}(T)$ for $j = m$. Arrows indicate edge direction from j to p_j. Here v_1, v_2, and v_3 are nodes 1, 2, and 11, respectively. T_1 and T_2 are outlined with dashed lines. The median of T_1 is node 6 because $\mathrm{w}(\mathrm{T}[6]) = 14 \geq \mathrm{w}(T_1)/2 = 28/2$ and the median of T_2 is node 5 with weight $15 \geq \mathrm{w}(T_2)/2 = 27/2$. When near medians are in $T_1 = \{1, 6, 7, 8, 9, 10\}$, the far medians are on the path $[0, 5]$. When they are in $\mathrm{T}[v_2] \cup \mathrm{T}[v_3] = (T \setminus T_1) \setminus m = \{2, 3, 4, 5, 11, 12\}$, they are on the path $[0, 6]$

An example of Lemma 6 arises when $S = T \setminus \mathrm{T}[6]$ and $S' = T \setminus \mathrm{T}[8]$. Here $\mathrm{w}(S) = 58 - 14 = 44$ and $\mathrm{w}(S') = 58 - 6 = 52$. Note that $S \setminus S' = \{8, 9, 10\}$ and $S' \setminus S = \{6\}$, so neither is contained in the other. Here $\mathrm{m}(S)$ is node 4 with $\mathrm{w}(\mathrm{T}[4]) = 22$ while $\mathrm{m}(S')$ is node 3 with $\mathrm{w}(\mathrm{T}[3]) = 26$. Put another way, the far median of node 6 is farther from m than that of node 8.

Now we have a context for describing the algorithms. To find the 2-medians of T we need only find the j such that

$$\mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], \mathrm{m}(T \setminus \mathrm{T}[j]))$$

is minimized. While doing so we also identify a *cut edge* $j\mathrm{p}_j$ that induces subtrees $\mathrm{T}[j]$ and $T \setminus \mathrm{T}[j]$. We locate the near and far median of j for each $j \in T$ and compute the costs. The latter is done incrementally, based on Lemmas 7 and 8 in Section 3.4 below.



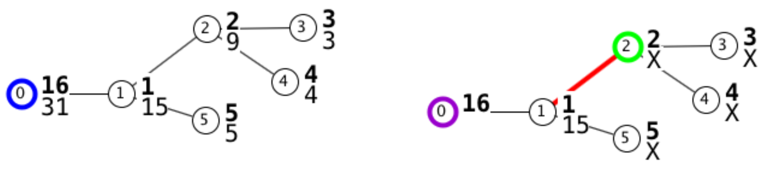
(a) 1-median on the left (b) the 2-medians for (a)

(c) 1-median on the right (d) the 2-medians for (c)

Fig. 4. A tree with two 1-medians.

3.3 Special cases

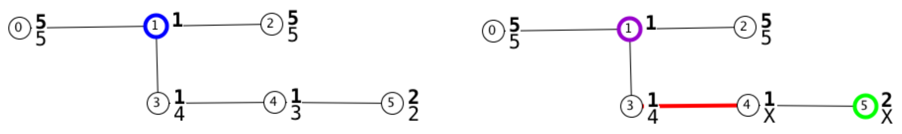
Fig. 4 illustrates a situation where there are two 1-medians as a result of having two equal-weight subtrees. The choice of 1-median is arbitrary in all of our definitions and algorithms. In Fig. 4(a) the 1-median (blue) is node 0 on the left, T_1 is the subtree $\{1, 4, 5\}$ on the right, and T_2 consists of single node 2. The trunk goes from node 2 to node 1, the other 1-median. Fig. 4(b) shows the optimum 2-median solution corresponding to (a): for cut edge 0,1 (red) in T_1 the near median in is node 1 (green) and far median is node 2 (red); total cost is 9. In Fig. 4(c) the 1-median is node 1 on the right, T_1 is the subtree $\{0, 2, 3\}$ and T_2 consists of single node 4. The trunk now includes one more node, node 4. However, it has weight less than that of the 1-median, so it is essentially a degenerate T_2 and its inclusion is a technicality. Now the near median for cut edge 2,0 in T_1 is node 2 and the far median (purple) is node 1, same as the 1-median. The medians are the same; only the cut edge differs. And this is only because the algorithm locating these medians started with near medians in T_1.



(a) the median with no T_2 (b) the 2-medians for (a)

Fig. 5. A tree with only one subtree after removal of the 1-median, i.e., no T_2.

Fig. 5 shows a tree that has a T_1 but no T_2 with respect to the 1-median, node 0. In this case one of the medians will always be the 1-median and the other will be the 1-median of T_1.



(a) the 1-median and weights of all subtrees (b) the 2-medians and the cut edge

Fig. 6. A tree whose 2-medians are not both on the trunk.

Lest the reader think that both 2-medians will always be on the trunk, the example in Fig. 6 illustrates a simple tree T where this is not the case. All edge lengths are 1. The trunk of T consists of nodes 0, 1, and 2 across the top of the illustration. Fig. 6(a) shows all the subtree weights assuming m (node 1) is the root. In this case T_1 and T_2 are interchangeable: they both consist of the single nodes of weight 5 (nodes 0 and 2). But, as shown in Fig. 6(b), the optimal solution has one of the medians in T_3, which consists of nodes 3, 4, and 5.

Any choice of 2-medians on the trunk includes either both 0 and 2 or $m = 1$ and one of 0 and 2. In the former case (using node 0 as the node closest to nodes 1, 3, 4, and 5), the cost is

$$1 \cdot \mathrm{d}[0,1] + 1 \cdot \mathrm{d}[0,3] + 1 \cdot \mathrm{d}[0,4] + 2 \cdot \mathrm{d}[0,5] = 1 + 2 + 3 + 8 = 14$$

in the latter (using node 0 as the median other than m) it is

$$5 \cdot \mathrm{d}[m,2] + 1 \cdot \mathrm{d}[m,3] + 1 \cdot \mathrm{d}[m,4] + 2 \cdot \mathrm{d}[m,5] = 5 + 1 + 2 + 6 = 14$$

The optimum solution shown in Fig. 6(b) has cost

$$5 \cdot \mathrm{d}[m,0] + 5 \cdot \mathrm{d}[m,2] + 1 \cdot \mathrm{d}[4,3] + 2 \cdot \mathrm{d}[4,5] = 5 + 5 + 1 + 2 = 13$$

3.4 Updating costs

We turn now to the facts that allow us to update costs incrementally as we traverse the tree to locate the medians.

The following gives the incremental cost when a cut edge is moved up toward m while keeping a near median candidate m^* the same. For now we assume the far median candidate stays at m, a feature of the algorithm described later.

Lemma 7. *If k is any child of j and m^* is in $\mathrm{T}[k]$ then*

$$\begin{aligned} &\mathrm{cost}(\mathrm{T}[j], m^*) + \mathrm{cost}(T \setminus \mathrm{T}[j], m) \\ &\quad - (\mathrm{cost}(\mathrm{T}[k], m^*) + \mathrm{cost}(T \setminus \mathrm{T}[k], m)) \\ &= (\mathrm{d}[m^*, j] - \mathrm{d}[m, j]) \cdot (\mathrm{w}(\mathrm{T}[j]) - \mathrm{w}(\mathrm{T}[k])) \end{aligned}$$

Proof. As shown in Fig. 7, $\mathrm{w}(\mathrm{T}[j]) - \mathrm{w}(\mathrm{T}[k])$ is the total weight of the nodes that switch sides, whose median is now m^* instead of m, while $\mathrm{d}[m^*, j] - \mathrm{d}[m, j]$ is the distance of those nodes from their new median minus that from their original one. □

Now consider the incremental cost of moving a median candidate closer to the root m; moving away from the root has the opposite effect.

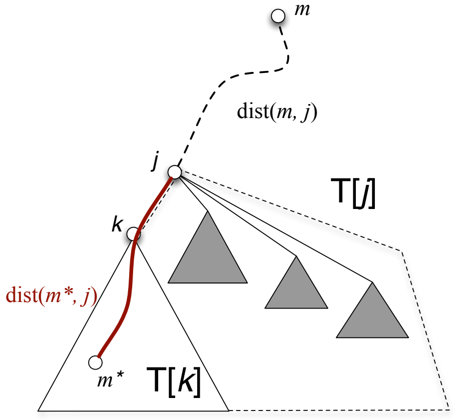


Fig. 7. Difference in cost as the cut edge moves closer to the root m.

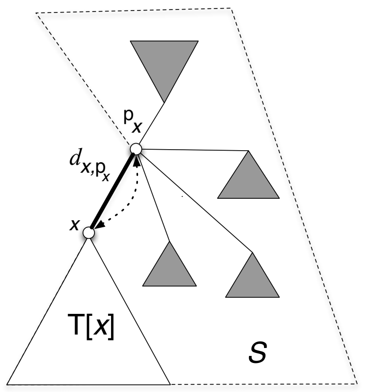


Fig. 8. Difference in cost as a median candidate moves closer to or farther from the root m.

Lemma 8. *If $x, \mathrm{p}_x \in S$ for some subtree S of T, then*

$$\mathrm{cost}(S, \mathrm{p}_x) - \mathrm{cost}(S, x) = d_{x,\mathrm{p}_x}(2\mathrm{w}(\mathrm{T}[x]) - \mathrm{w}(\mathrm{T}[S]))$$

Proof. Note that nodes in $\mathrm{T}[x]$ move d_{j,p_j} farther away from the candidate median while nodes in $S \setminus \mathrm{T}[x]$ move closer by the same amount. So the difference is

$$d_{x,\mathrm{p}_x}\mathrm{w}(\mathrm{T}[x]) - d_{x,\mathrm{p}_x}(\mathrm{w}(S) - \mathrm{w}(\mathrm{T}[x])) = d_{x,\mathrm{p}_x}(2\mathrm{w}(\mathrm{T}[x]) - \mathrm{w}(S))$$

See Fig. 8. □

The concepts of near and far median, by different names, and the cost updates also appear in Breton's thesis [3].

4 Linear-Time Computation of Near Medians

Before we introduce the algorithm for computing near medians, we define some notation specific to our algorithms.

After m, a *1-median* of T, has been identified we can, as pointed out earlier, use it as the root of T and use it define a parent p_j for each node j; if $j = m$ then p_j is undefined. Other definitions follow naturally.

- an *ancestor* of j is any node on the path $[j, m]$.
- if i is p_j, then j is a *child* of i;
- $\mathrm{T}[j]$ is the subtree of T rooted at j; it can be defined recursively by $j \in \mathrm{T}[j]$ and, if $\mathrm{p}_k \in \mathrm{T}[j]$, then $k \in \mathrm{T}[j]$;
- for convenience we use $\mathrm{m}(j)$ to denote $\mathrm{m}(\mathrm{T}[j])$ and $\mathrm{m}'(j)$ to denote $\mathrm{m}(T \setminus \mathrm{T}[j])$, the near and far medians of j, respectively;
- we use $\mathrm{cost}[j]$ to denote $\mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], \mathrm{m}'(j))$, the cost of a solution based on use of $j\mathrm{p}_j$ as cut edge and the medians of $\mathrm{T}[j]$ and $T \setminus \mathrm{T}[j]$.

Goldman [6] proved that the one-median m can be computed in linear time, i.e., $\mathrm{O}(n)$, where n is the number of tree nodes. It is also easy to compute in linear time, for each node j, the quantities $\mathrm{w}(\mathrm{T}[j])$ and $\mathrm{d}[j, m]$, the former bottom up (and implicitly during Goldman's algorithm), the latter top down recursively: $\mathrm{d}[m, m] = 0$, and, for $j \neq m$, $\mathrm{d}[j, m] = d_{j,\mathrm{p}_j} + \mathrm{d}[\mathrm{p}_j, m]$. Computing $\mathrm{cost}(T, m)$ follows directly. Identifying T_1 and T_2 (if it exists) is easy; and, using Goldman's algorithm again, we can compute $\mathrm{m}(T')$ for $T' = T_1$ and $T' = T_2$.

The linear time algorithm for near medians is outlined in Fig. 9. The algorithms described in Sections 5–7 differ only in their approach to computing the far median (candidates), as encapsulated by the GETFARMEDIAN function in line 10.

An easy way to ensure that node j is processed only after all its children have been processed – line 1, is to use a queue Q. Initially Q contains the leaves in $\hat{T}$. We keep track of the *residual degree* of each node k, i.e., the number of children that have not been processed, initially $\deg(k) - 1$. When this reaches 0 we add k to Q.

The algorithm does incremental cost updates using Lemmas 7 and 8.

```
procedure computeNearMedians is
    $\hat{T}$ is either $T_1$ or $(T \setminus T_1) \setminus m$
    Computes $\mathrm{m}(j)$ and the corresponding cost:
        $\mathrm{cost_m}[j] = \mathrm{cost}(\mathrm{T}[j], \mathrm{m}(\mathrm{T}[j])) + \mathrm{cost}(T \setminus \mathrm{T}[j], m)$
    for every $j$ in $\hat{T}$.
    This means every possible cut edge $(j, \mathrm{p}_j)$ is explored.
    $bestCost$ is initially $\infty$ and updated every execution of line 10.
    do until all nodes $j \in \hat{T}$ have been processed
        (1) let $j$ be a node all of whose children have been processed
        ▷ processing $j$; the edge $(j, \mathrm{p}_j)$ is being considered as a cut edge
        if $\deg(j) = 1$ then    ▷ $j$ is a leaf and the only node in its subtree
            (2) $\mathrm{m}(j) \leftarrow j$
            (3) $cost \leftarrow \mathrm{cost}(T, m) - \mathrm{d}[j, m] \cdot w_j$
        else
            (4) let $k^*$ be the child of $j$ with maximum $\mathrm{w}(\mathrm{T}[k^*])$
            (5) find $\mathrm{m}(j)$ on the path $[\mathrm{m}(k^*), j]$
            (6) $cost \leftarrow \mathrm{cost_m}[k^*]$
            ▷ update $cost$ using Lemmas 7 and 8
            (7) $cost \leftarrow cost + (\mathrm{d}[\mathrm{m}(k^*), j] - \mathrm{d}[m, j]) \cdot (\mathrm{w}(\mathrm{T}[j]) - \mathrm{w}(\mathrm{T}[k^*]))$
            for each edge $(x, p_x)$ on the path $[\mathrm{m}(k^*), \mathrm{m}(j)]$ do
                (8) $cost \leftarrow d_{x,\mathrm{p}_x}(2\mathrm{w}(\mathrm{T}[x]) - \mathrm{w}(\mathrm{T}[j]))$
            end do
            (9) $\mathrm{cost_m}[j] \leftarrow cost$
            (10) $bestCost \leftarrow$ updateCost($j$, getFarMedian($j$), $bestCost$)
        endif
    end do    ▷ until all nodes have been processed
end ComputeMedians
```

Fig. 9. Linear time algorithm for computing all near medians.

```
procedure computeDeltas is
    ▷ assumes $\mathrm{w}(\mathrm{T}[j])$ has been computed for all $j \in P^*$,
    ▷       where $P^*$ is a segment of the trunk starting at $m$
    for node $current$ on path $P^*$ (starting with $m$) do
        if $current = m$ then $\Delta(current) \leftarrow 0$
        else $\Delta(current) \leftarrow \Delta(\mathrm{p}_{current}) + \mathrm{d}[current, \mathrm{p}_{current}] \cdot (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[current]))$ endif
    end do
end computeDeltas
```

Fig. 10. Algorithm for computing Δ values along path P^*.

```
function UPDATECOST($j, m', c$) is
    Assumes $\mathrm{cost_m}[j]$ and $\mathrm{w}(\mathrm{T}[j])$ have been computed for all $j \in \hat{T}$, and that $\Delta(i)$ has
    been computed for $i \in P^*$. Computes
        $totalCost = \mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], m')$
    returns $totalCost$ if $totalCost < c$ and $c$ otherwise.

    $totalCost \leftarrow \mathrm{cost_m}[j] + \Delta(m') - \mathrm{d}[m', m] \cdot \mathrm{w}(\mathrm{T}[j])$
    if $totalCost < c$ then return $totalCost$ else return $c$ endif
end UPDATECOST
```

Fig. 11. Algorithm for computing total cost given a far median candidate.

By Lemma 4, the nodes on the path $[\mathrm{m}(k^*), j)$ are never accessed again after an execution of line 5 for node j. Future searches for $\mathrm{m}(i)$, where i is an ancestor of j, will begin at $\mathrm{m}(j)$ or one of its ancestors. Thus, each node except for $\mathrm{m}(j)$ is accessed once in line 5, and $\mathrm{m}(j)$ is accessed once for each j. The total number of accesses to nodes in lines 5 and 8 is therefore $\mathrm{O}(n)$.

For each node $j \neq m$ we keep track of $\mathrm{cost_m}[j] = \mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], m)$. The far median $\mathrm{m}(T \setminus \mathrm{T}[j])$ will not necessarily be m, but we use this as a place holder until the correct far median is computed. The appropriate adjustment is made using $\Delta(\mathrm{m}(T \setminus \mathrm{T}[j]))$, defined below and in procedure COMPUTEDELTAS – see Fig. 10. Because all children of j are processed before j, we already know $\mathrm{cost_m}[k^*]$, where k^* is the child whose subtree has maximum weight – see line 4. Lemmas 7 and 8 are then used to update the cost from $\mathrm{cost_m}[k^*]$ to $\mathrm{cost_m}[j]$ – lines 6, 7, and 8.

Note that $\mathrm{d}[k^*, j] = \mathrm{d}[m, k^*] - \mathrm{d}[m, j]$ in line 7 can be computed directly using precomputed distances from m.

To compute the cost of using a specific far median candidate m', i.e., $\mathrm{cost}(T \setminus \mathrm{T}[j], m')$, we precompute $\Delta(m') = \mathrm{cost}(T, m) - \mathrm{cost}(T, m')$ for each m' on the trunk. Starting with $\Delta(m, m) = 0$ we use Lemma 8:

$$\begin{aligned} &\Delta(m'),\ \text{when } m' \neq m \\ &= \Delta(\mathrm{p}_{m'}) + d_{m',\mathrm{p}_{m'}}(\mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[m'])) \end{aligned}$$

These values can be computed top down starting with the two m' candidates that have $\mathrm{p}_{m'} = m$. When a far median candidate $\mathrm{m}'(j)$ is identified, we can compute $\mathrm{cost}[j]$ from $\mathrm{cost_m}[j]$ as follows. First observe that

$$\mathrm{cost}[j] = \mathrm{cost_m}[j] + \mathrm{cost}(T \setminus \mathrm{T}[j], \mathrm{m}'(j)) - \mathrm{cost}(T \setminus \mathrm{T}[j], m)$$

Then, the difference between previously computed cost and true cost is

$$\begin{aligned} &\mathrm{cost}(T \setminus \mathrm{T}[j], \mathrm{m}'(j)) - \mathrm{cost}(T \setminus \mathrm{T}[j], m) \\ &= (\mathrm{cost}(T, \mathrm{m}'(j)) - \mathrm{d}[j, \mathrm{m}'(j)] \cdot \mathrm{w}(\mathrm{T}[j])) - (\mathrm{cost}(T, m) - \mathrm{d}[j, m] \cdot \mathrm{w}(\mathrm{T}[j])) \\ &= (\mathrm{cost}(T, \mathrm{m}'(j)) - \mathrm{cost}(T, m)) + (\mathrm{d}[j, m] - \mathrm{d}[j, \mathrm{m}'(j)]) \cdot \mathrm{w}(\mathrm{T}[j]) \\ &= \Delta(\mathrm{m}'(j)) - \mathrm{d}[\mathrm{m}'(j), m] \cdot \mathrm{w}(\mathrm{T}[j]) \end{aligned}$$

The algorithms in Figs. 10 and 11 summarize the computation of Δ values and the use of these when a far median candidate has been identified, respectively.

5 An Algorithm Based on Sorting by Subtree Weights

One approach, originally proposed by Hämäläinen [9], takes advantage of Lemma 6. If nodes j are processed (line 1) in order of nondecreasing subtree weight $\mathrm{w}(\mathrm{T}[j])$, consider two nodes k and j, processed in two consecutive iterations of the main loop. Lemma 6 says that $\mathrm{m}'(k)$ is on the path $[m, \mathrm{m}'(j)]$, meaning that nodes on the path $[m, \mathrm{m}'(k))$ need no longer be considered in future iterations and the starting point in the search for $\mathrm{m}'(j)$ is $\mathrm{m}'(k)$. The time bound analysis for finding all the far medians is analogous to that for line 5 in the near median algorithm – Fig. 9. To identify $\mathrm{m}'(j)$, i.e., implement getFarMedian, do the following:

```
while $x \in [\mathrm{m}'(k), \mathrm{m}(T')]$ (moving in the implied direction)
        and $\mathrm{w}(\mathrm{T}[x]) \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$ do
    $last \leftarrow x$
end do
return $last$
```

We can modify the near-median algorithm so that nodes j are processed in monotonic nondecreasing values of $\mathrm{w}(\mathrm{T}[j])$ in line 1. If all w_j are positive, simply sort the nodes by increasing subtree weight $\mathrm{w}(\mathrm{T}[j])$ and process them in sorted order. If some node weights are 0,[3] break ties by putting nodes that are farther away from m (the path has more edges) earlier in the order.

The sort takes $\mathrm{O}(n \lg n)$ using a standard sorting algorithm. Radix sort yields a time bound of $\mathrm{O}(n \lg w_{\max} / \lg n)$, where $w_{\max}$ is the largest subtree weight, an improvement to linear time if the subtree weights are polynomial in n.[4]

Another alternative is to use a priority queue Q in place of the ordinary queue proposed earlier, adding a node j to Q when all of its children have been processed and using $\mathrm{w}(\mathrm{T}[j])$ as its key. Initially the leaves of $\hat{T}$ are added to Q. The next node to be processed in the main loop is one with minimum $\mathrm{w}(\mathrm{T}[j])$. The time for this implementation is $\mathrm{O}(n \lg n_L)$, where n_L is the number of leaves in the tree, a bound on the number of nodes that can be in the queue at any given time, an improvement if the tree consists of a small number of long paths.[5]

6 An Algorithm using Binary Search

An algorithm proposed by Gavish and Sridhar [5] locates each far median using binary search. See Fig. 12. After each iteration of the **while** loop, the number of edges on the path $[near, far]$ is cut in half. If it is originally odd, say ℓ, the new length is either $(\ell - 1)/2$ or $(\ell + 1)/2$. This means that the number of comparisons between $\mathrm{w}(\mathrm{T}[m'])$, where m' is a candidate for the far median, and $(\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$ is $\mathrm{O}(\lg n_{\mathrm{P}})$, where n_{P} is the number of edges on the path being searched.

This algorithm therefore has a runtime of $\mathrm{O}(n \lg(\max(\ell_1, \ell_2)))$, where ℓ_1, ℓ_2 are the numbers of edges in paths $[m, \mathrm{m}(T_1)]$ and $[m, \mathrm{m}(T_2)]$, respectively. Recall that these paths form the trunk, defined earlier in Section 3, and note that they are easy to identify after the 1-median has been computed, T_1 and T_2 have been identified, and their medians computed – the preliminary steps of the near

[3] If $w_{\mathrm{p}_j} = 0$ then $\mathrm{w}(\mathrm{T}[\mathrm{p}_j]) = \mathrm{w}(\mathrm{T}[j])$.

[4] Breton [3] points out that the 2-medians can be found in linear time if node weights can be sorted in linear time.

[5] A recent MS Thesis by Acharyya [1] achieves the same result using techniques of computational geometry.

```
function BinarySearch($j$) is
    ▷ returns the median of $T \setminus \mathrm{T}[j]$, as specified in Lemma 5.
    ▷ assumes that the nodes on path $[m, \mathrm{m}(T')]$ are stored in an array
    ▷ at any stage path is a subpath of $[m, \mathrm{m}(T')]$
    path ← $[near, far]$, where $near = m$ and $far = \mathrm{m}(T')]$
    ▷ Invariant: $[near, far]$ contains a node $k$ for which
    ▷     $\mathrm{w}(\mathrm{T}[k]) \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$, and,
    ▷     for every $x \in (k, far]$, $\mathrm{w}(\mathrm{T}[x]) < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$
    while path has more than one node do
        mid ← midpoint of path, closer to far if there's a tie
        if $\mathrm{w}(\mathrm{T}[mid]) \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$ then
            path ← $[mid, far]$
        else
            path ← $[near, mid)$
        endif
    end do
    return near
end BinarySearch
```

Fig. 12. Binary search to locate $\mathrm{m}(T \setminus \mathrm{T}[j])$ on the path $[m, \mathrm{m}(T')]$, where T' is one of T_1 or T_2.

median algorithm. This means that a short trunk will be recognized before a decision is made about which of our algorithms to run.

7 An Algorithm Based on Distances

From Lemma 2 we know that, if $j\mathrm{p}_j$ is the cut edge for an *optimum* solution, then one of the following holds:

$$\begin{aligned}
\mathrm{d}[\mathrm{m}(j), j] &= \mathrm{d}[j, \mathrm{m}'(j)] && \text{the midpoint between the near and far medians is at } j \\
\mathrm{d}[\mathrm{m}(j), \mathrm{p}_j] &= \mathrm{d}[\mathrm{p}_j, \mathrm{m}'(j)] && \text{the midpoint is at } \mathrm{p}_j \\
\mathrm{d}[\mathrm{m}(j), j] + \delta &= \mathrm{d}[\mathrm{p}_j, \mathrm{m}'(j)] && \text{where } \delta < d_{j,\mathrm{p}_j}\text{, i.e., the midpoint lies somewhere on the edge } j\mathrm{p}_j
\end{aligned}$$

Each of the three cases identifies a candidate location x for $\mathrm{m}'(j)$. If x is not the true far median, it does not matter. In that case

$$\begin{aligned}
&\mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], x) \\
&\geq \mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], \mathrm{m}'(j)) \\
&= \mathrm{cost}[j] \geq \mathrm{cost}[j^*] \text{ (the optimum)}
\end{aligned}$$

We can go ahead and compute the total cost using x as a candidate far median without fear of incorrectly identifying the optimum solution. We say that cut edge $j\mathrm{p}_j$ is *dominated* if it can be ruled out as a cut edge for the true 2-medians m_1 and m_2.

Suppose that all edge lengths $d_{i,j} = 1$. Then the only possible value for δ in the last case is $1/2$, which implies that $\mathrm{d}[\mathrm{m}(j), j] = \mathrm{d}[\mathrm{p}_j, \mathrm{m}'(j)]$. So only three candidate locations need to be considered.

The algorithm for each $\hat{T}$, where $\hat{T} = T_1$ or $\hat{T} = (T \setminus T_1) \setminus m$ is straightforward. Here T' is either T_2 or T_1, respectively, making $[m, T']$ the relevant segment of the trunk.

1. put nodes $m' \in [m, \mathrm{m}(T')]$ consecutively into an array $A[0] = m, \ldots, A[\ell] = \mathrm{m}(T')$, where $\ell = \mathrm{d}[m, \mathrm{m}(T')]$.
2. for each node $j \in \hat{T}$, compute three quantities:
   (a) $d_1 = \mathrm{d}[\mathrm{m}(j), \mathrm{p}_j] - \mathrm{d}[\mathrm{p}_j, m]$
   (b) $d_2 = \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[\mathrm{p}_j, m]$
   (c) $d_3 = \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m]$
   for $i = 1, 2, 3$ do

   if $0 \le d_i \le \ell$ then $\mathit{bestCost} \leftarrow \textsc{updateCost}(\mathit{bestCost}, \mathrm{cost_m}[j], A[d_i])$

So, effectively, line 10 of the near median algorithm is expanded into three options. [6]

This idea can be generalized to arbitrary integer edge lengths to obtain an algorithm with $\mathrm{O}(nd_{\max})$ runtime, where $d_{\max}$ is the maximum length of an edge. With integer edge lengths, Lemma 2 tells us that the midpoint between the near and far median will always be at a position that is an integer multiple of $\frac{1}{2}$. So for the cut edge (j, p_j) with length d_{j,p_j}, the midpoint between a near and far median (if that cut edge is not *dominated*) could be any j, p_j, or any distance δ with $0 < \delta < d_{j,\mathrm{p}_j}$ from j, where 2δ is an integer. That leads to $2d_{j,\mathrm{p}_j} + 1$ options.

Not all options correspond to actual nodes along the path. So we need to initialize the array A as follows.

- create A so that it has entries $A[0], \ldots, A[d']$, where $d' = \mathrm{d}[m, \mathrm{m}(T')]$
- initialize all of these entries to **null**.
- for each node x on the path $[m, \mathrm{m}(T')]$, let $A[\mathrm{d}[m, x]] = x$

The array A will have some gaps. The total number of entries is bounded by $nd_{\max}$, where $d_{\max}$ is the maximum length of any edge.

The more general algorithm for finding far median candidates and updating costs, for each cut edge (j, p_j), is as follows.

```
for $\delta \leftarrow 0$ to $d_{j,\mathrm{p}_j}$, by steps of $\frac{1}{2}$ do
    let $d = \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m] + 2\delta$
    if $0 \le d \le d'$ and $A[d] \ne$ null then
        $\mathit{bestCost} \leftarrow \textsc{updateCost}(\mathit{bestCost}, \mathrm{cost_m}[j], A[d])$
    endif
end do
```

Since d' is known in advance of the execution of the loop, we need only iterate over δ values ranging from $(\mathrm{d}[j, m] - \mathrm{d}[\mathrm{m}(j), j])/2$ to the minimum of $(\mathrm{d}[j, m] - \mathrm{d}[\mathrm{m}(j), j] + d')/2$ and d_{j,p_j}.

In the above-described algorithm for unit edge lengths,

$$\begin{aligned} d_1 &= \mathrm{d}[\mathrm{m}(j), \mathrm{p}_j] - \mathrm{d}[\mathrm{p}_j, m] = (\mathrm{d}[\mathrm{m}(j), j] + 1) - (\mathrm{d}[j, m] - 1) \\ &= \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m] + 2 && \text{so } \delta = 1 \\ d_2 &= \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[\mathrm{p}_j, m] = \mathrm{d}[\mathrm{m}(j), j] - (\mathrm{d}[j, m] - 1) = \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m] + 1 && \text{so } \delta = \tfrac{1}{2} \\ d_3 &= \mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[\mathrm{p}_j, m] && \text{so } \delta = 0 \end{aligned}$$

[6] Breton [3] reports a similar linear time algorithm for the case of unit edge lengths.

8 Conclusions

The algorithms we have considered all rely on the fact that *near medians can be computed in linear time*. The ones based on sorting by subtree weights and on binary search correctly identify far medians along the way. In the first case, sorting by weights, this is achieved using Lemma 6 to guarantee that there is no backtracking as we search along the trunk. In the second, binary search is used to find the far median.

The algorithm based on distances uses Lemma 2 to rule out far median candidates that are dominated, i.e., they either cannot be correct far medians for the cut edge $j\mathrm{p}_j$ or the cut edge is *not* one corresponding to an optimal solution. In the process, the algorithm may explore candidates that are not correct far medians and compute their costs.

Based on the algorithms described here, the runtime for finding the two-medians of a tree can be improved to better than $\mathrm{O}(n \lg n)$ in four situations, all of which can be detected initially or after the one-median is computed:

1. When the node weights are small, runtime for an algorithm that sorts by subtree weight (see Section 5) is $\mathrm{O}(n \frac{\lg w_{\max}}{\lg n})$, where $w_{\max}$ is the largest node weight, which is linear if node weights are polynomial.
2. When T has only a small number n_L of leaves, runtime for the subtree-weight algorithm using a priority queue is $\mathrm{O}(n \lg n_L)$.
3. When the lengths (number of edges) of paths $[m, \mathrm{m}(T_1)]$ and $[m, \mathrm{m}(T_2)]$ are both small, runtime for an algorithm that uses binary search to locate the far medians is $\mathrm{O}(n \lg \ell)$, where ℓ is the length of the longer of these two paths.
4. When all edge lengths are $\leq d_{\max}$, runtime is $\mathrm{O}(nd_{\max})$, which is linear if edge lengths are bounded by a constant. In this case $d_{\max}$ is a bound on the number of far median candidates that need to be considered for each cut edge.

Our near median algorithm acts as a framework for all of these improvements. Future algorithms can either improve the efficiency of searches for far median candidates during their execution or compute the optimal solution *after all near medians are known.* The algorithms reported here run in linear time, i.e., $\mathrm{O}(n)$, when

1. node weights are polynomial in n;
2. the number of leaves in T is bounded by a constant;
3. the number of nodes on the path between medians of the two largest subtrees with respect to m is bounded by a constant; or
4. the edge lengths are bounded by a constant;

Fig 13 illustrates a situation where our improvements do not reduce the time bound. The figure shows only T_1 but T_2 is identical; here $n = 4k + 1$.

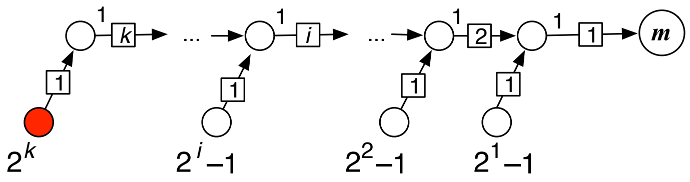


Fig. 13. An example for which none of our improvements apply. Only one of two subtrees is shown. Node weights are next to the nodes. Edge lengths are in the squares along the edges.

1. Node weights range from 1 to $w_{\max} = 2^k$, which is exponential in n, so the first improvement does not apply.
2. There are $2k$ leaves, so the second improvement fails as well.
3. The node farthest from m in each subtree is the median of that subtree: the total weight of the subtree is $\sum_{1 \leq i \leq k} 2^i + 1 = 2^{k+1}$ and the weight of the last node is half of that. The third case is therefore also ruled out.
4. Edge lengths range from 1 to $d_{\max} = k$, ruling out the third case. Even if we are careful to restrict δ values as suggested in Section 7, there are still too many nodes with a non-constant number of options. Consider, for example, the $k/2$ nodes in the top row closest to m; refer to them as nodes $j = 1, \dots, k/2$, moving away from m (right to left). For each j we have

$$\begin{aligned}
&\mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m] \\
&= \Big(\sum_{1 \leq i \leq k+1} i - \sum_{1 \leq i \leq j} i\Big) - \sum_{1 \leq i \leq j} i \\
&= \sum_{1 \leq i \leq k+1} i - 2 \cdot \sum_{1 \leq i \leq j} i \\
&\approx k^2/2 - j^2
\end{aligned}$$

So, for each such j, δ can range from 0 to roughly $(2k^2 - j^2)/2$. Summing these up leads to $\Omega(k^2)$ total options considered, which is $\Omega(n^2)$.

References

1. R. Acharyya, *2-median problems in tree networks*, Master's thesis, Simon Frazier University, March 2017. http://summit.sfu.ca/item/17232. 14
2. R. Benkoczi and B. Bhattacharya, *A new template for solving p-median problems for trees in sub-quadratic time*, in Proc. European Symposium on Algorithms, no. 3669 in Lecture Notes in Computer Science, 2005, pp. 271–282. 1, 2
3. D. Breton, *Facility location problems in trees*, Master's thesis, Simon Frazier University, November 2002. 2, 11, 14, 16
4. P. M. Dearing, R. L. Francis, and T. J. Lowe, *Convex location problems in tree networks*, Operations Research, 24 (1976), pp. 628–642. 2
5. B. Gavish and S. Sridhar, *Computing the 2-median on tree networks in $O(n \log n)$ time*, Networks, 26 (1995), pp. 305–317. 1, 2, 14
6. A. J. Goldman, *Optimal center location in simple networks*, Transportation Science, 5 (1971), pp. 212–221. 1, 2, 4, 11

7. S. L. Hakimi, *Optimum locations of switching centers and the absolute centers and medians of a graph.*, Operations Research, 12 (1964), pp. 450–459. https://doi.org/10.1287/opre.12.3.450. 1
8. ———, *Optimum distribution of switching centers in a communication network and some related graph theoretic problems*, Operations Research, 13 (1965), pp. 462–475. https://doi.org/10.1287/opre.13.3.462. 1
9. P. Hämäläinen, *An $O(n \log n)$ implementation of the edge deletion method to find a 2-median in a tree network*. unpublished, 1988. 1, 2, 14
10. O. Kariv and S. L. Hakimi, *An algorithmic approach to network location problems. II: the p-medians*, SIAM J. Appl. Math., 37 (1979), pp. 539–560. 1, 2
11. P. Mirchandani and A. Oudjit, *Localizing 2-medians on probabilistic and deterministic tree networks*, Networks, 10 (1980), pp. 329–350. 1, 2
12. N. Mladenovic, J. Brimberg, P. Hansen, and J. A. Moreno-Perez, *The p-median problem: A survey of metaheuristic approaches*, European Journal of Operational Research, 179 (2007), pp. 927–939. 2
13. J. Reese, *Solution methods for the p-median problem: An annotated bibliography*, Networks, 48 (2006), pp. 125–142. 2
14. C. S. ReVelle and R. W. Swain, *Central facilities location*, Geographical Analysis, 2 (1970), pp. 30–42. 2
15. M. Stallmann, *Algorithm animation with Galant*, IEEE Computer Graphics and Applications, 37 (2017), pp. 8–14. 20
16. A. Tamir, *An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs*, Operations Research Letters, 19 (1996), pp. 59–64. 1, 2

Appendix

We now examine the behavior of the algorithms on two examples with unit-length edges and one more example with edges of arbitrary length.[7] In each case, we first show the computation of the near medians $\mathrm{m}(j)$ and $\mathrm{cost_m}[j]$ as described in the near median algorithm. We also show the computation of the Δ values. Finally, we describe the identification of far medians (or candidate far medians in the case of the distance-based algorithm) and the computations of $\mathrm{cost}[j]$ for each j (keeping track of *bestCost* as we go).

A Algorithms on the Example of Fig. 3

After we compute the one-median $m = 0$, the distance $\mathrm{d}[j, m]$ is simply the number of edges on the path from j to m. The subtree weights, shown in Fig. 14, are computed by Goldman's algorithm. We also precompute $\mathrm{cost}(T, m) = 153$. Furthermore we compute $\mathrm{m}(T')$ for $T' = T_1$ and T_2: in this case $\mathrm{m}(1)$ is node 6 and $\mathrm{m}(2)$ is node 5.

Next we compute Δ values for nodes along the paths $(0, 6]$ and $(0, 5]$

$$\Delta(1) = \Delta(0) + \mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[1]) = 0 + 58 - 56 = 2$$
$$\Delta(6) = \Delta(1) + \mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[6]) = 2 + 58 - 28 = 32$$

$$\Delta(2) = \Delta(0) + \mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[2]) = 0 + 58 - 54 = 4$$
$$\Delta(3) = \Delta(2) + \mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[3]) = 4 + 58 - 52 = 10$$
$$\Delta(4) = \Delta(3) + \mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[4]) = 10 + 58 - 44 = 24$$
$$\Delta(5) = \Delta(4) + \mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[5]) = 24 + 58 - 30 = 52$$

The near median and $\mathrm{cost_m}[j]$ computations are shown in Tables 1 and 2.

Table 1. Calculation of $\mathrm{cost_m}[j]$ for $j \in T_1$ in the example of Fig. 14.

j	p_j	k^*	$\mathrm{w}(\mathrm{T}[j])$	$\mathrm{d}[\mathrm{m}(k^*), j] - \mathrm{d}[m, j]$	$\mathrm{w}(\mathrm{T}[j]) - \mathrm{w}(\mathrm{T}[k^*])$	$\mathrm{cost_m}[j]$
10	8	N/A	1	$\mathrm{d}[j, m] = 4$		$153 - 4 \cdot 1 = 149$
9	8	N/A	2	$\mathrm{d}[j, m] = 4$		$153 - 4 \cdot 2 = 145$
14	1	N/A	14	$\mathrm{d}[j, m] = 2$		$153 - 2 \cdot 14 = 125$
8	7	9	6	$1 - 3 = -2$	$6 - 2 = 4$	$-8 \ldots$
… (median moves to 8)				$\mathrm{d}[k^*, j] \cdot (2\mathrm{w}(\mathrm{T}[k^*]) - \mathrm{w}(\mathrm{T}[j])) = 1 \cdot (4 - 6) = -2$		$145 - 8 - 2 = 135$
7	1	8	9	$1 - 2 = -1$	$9 - 6 = 3$	$135 - 1 \cdot 3 = 132$
1	0	6	28	$1 - 1 = 0$	$28 - 14 = 14$	$125 - 0 \cdot 14 = 125$

[7] The pictures in the figures were generated using an animation of the subtree weight sorting algorithm. The animation was created using Galant [15], a general purpose graph algorithm animation tool. Source code for this specific animation and source files for the examples are available on request. The animation tool itself is at `https://github.com/mfms-ncsu/galant`.

Table 2. Calculation of $\mathrm{cost_m}[j]$ for $j \in (T \setminus T_1) \setminus m$ in the example of Fig. 14.

j	p_j	k^*	$\mathrm{w(T}[j])$	$\mathrm{d}[\mathrm{m}(k^*), j] - \mathrm{d}[m, j]$	$\mathrm{w(T}[j]) - \mathrm{w(T}[k^*])$	$\mathrm{cost_m}[j]$
12	11	N/A	1	$\mathrm{d}[j, m] = 2$		$153 - 2 \cdot 1 = 151$
5	4	N/A	15	$\mathrm{d}[j, m] = 4$		$153 - 4 \cdot 15 = 93$
11	0	12	2	$1 - 1 = 0$	$2 - 1 = 1$	$151 + 0 \cdot 1 = 151$
4	3	5	22	$1 - 3 = -2$	$22 - 15 = 7$	$93 - 2 \cdot 7 = 79$
3	2	5	26	$2 - 2 = 0$	$26 - 22 = 4$	$79 + 0 \cdot 4 = 79$
2	0	5	27	$3 - 1 = 2$	$27 - 26 = 1$	$79 + 2 \cdot 1 = 81$

Sorting by weights on the example of Fig. 3

Figs. 15 through 21 show most of the steps steps taken during an algorithm based on sorting by subtree weights.

The nodes are sorted by nondecreasing subtree weights, so the order, with weights in parentheses is

$$10(1),\quad 9(2),\quad 8(6),\quad 7(9),\quad 6(14),\quad 1(28)$$

Fig. 15 shows the state of affairs when node 10 is processed. Since $\mathrm{w}(T) - \mathrm{w(T}[10]) = 57$ and $\mathrm{w(T}[2]) = 27 < 57/2$, $\mathrm{m}'(10) = 0$.

$$\mathrm{cost}[10] = \mathrm{cost_m}[10] = 149$$

When node 9 is processed (not shown), $\mathrm{w}(T) - \mathrm{w(T}[9]) = 56$ and $\mathrm{w(T}[2]) = 27 < 56/2$, so $\mathrm{m}'(9)$ is still 0 and

$$\mathrm{cost}[9] = \mathrm{cost_m}[9] = 145$$

Node 8 is next, see Fig. 16. Here $\mathrm{w}(T) - \mathrm{w(T}[8]) = 52$; $\mathrm{w(T}[2]) = 27$ and $\mathrm{w(T}[3]) = 26$, both $\geq 52/2$; but $\mathrm{w(T}[4]) = 22 < 52/2$; so $\mathrm{m}'(8) = 3$.

$$\mathrm{cost}[8] = \mathrm{cost_m}[8] + \mathit{\Delta}(3) - \mathrm{d}[3, m] \cdot \mathrm{w(T}[8]) = 135 + 10 - 2 \cdot 6 = 133$$

Node 7 (not shown) is next: the near median stays at node 8. For the far median, $\mathrm{w}(T) - \mathrm{w(T}[7]) = 58 - 9 = 49$ and $\mathrm{w(T}[4]) = 22 < 49/2$ so $\mathrm{m}'(7)$ stays at node 3.

$$\mathrm{cost}[7] = \mathrm{cost_m}[7] + \mathit{\Delta}(3) - \mathrm{d}[3, m] \cdot \mathrm{w(T}[7]) = 132 + 10 - 2 \cdot 9 = 124$$

Node 6 causes the far median to move again, to node 4 – see Fig. 17.

$$\mathrm{cost}[6] = \mathrm{cost_m}[6] + \mathit{\Delta}(4) - \mathrm{d}[4, m] \cdot \mathrm{w(T}[6]) = 125 + 24 - 3 \cdot 14 = 107$$

Node 1 is the last to be added for $\hat{T} = T_1$ – see Fig. 18. The far median moves to node 5.

$$\mathrm{cost}[1] = \mathrm{cost_m}[1] + \mathit{\Delta}(5) - \mathrm{d}[5, m] \cdot \mathrm{w(T}[1]) = 125 + 52 - 4 \cdot 28 = 65$$

The cut edge (1,0) with near median 6 and far median 5 yields the best solution so far.

Now we apply the algorithm again with $\hat{T} = (T \setminus T_1) \setminus m$. The nodes, in sorted order, with subtree weights in parentheses, are

$$12(1), \quad 11(2), \quad 5(15), \quad 4(22), \quad 3(26), \quad 2(27)$$

Fig. 19 shows the consideration of node 12 with cut edge (12,11). The far median is $m = 0$ and

$$\mathrm{cost}[12] = \mathrm{cost_m}[12] = 151$$

Node 11 is next, see Fig 20. Here the far median moves to node 1.

$$\mathrm{cost}[11] = \mathrm{cost_m}[11] + \varDelta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[11]) = 151 + 2 - 1 \cdot 2 = 151$$

The far median stays at node 1 for the remaining iterations. Fig. 21 shows the cut edge giving minimum cost. The remaining cost computations are as follows.

$$\begin{aligned}
\mathrm{cost}[5] &= \mathrm{cost_m}[5] + \varDelta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[5]) = 93 + 2 - 1 \cdot 15 = 80 \\
\mathrm{cost}[4] &= \mathrm{cost_m}[4] + \varDelta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[4]) = 79 + 2 - 1 \cdot 22 = 59 \\
\mathrm{cost}[3] &= \mathrm{cost_m}[3] + \varDelta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[3]) = 79 + 2 - 1 \cdot 26 = 55 \\
\mathrm{cost}[2] &= \mathrm{cost_m}[2] + \varDelta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[2]) = 81 + 2 - 1 \cdot 27 = 56
\end{aligned}$$

Thus, node 3 with cut edge (3,2) with near and far medians at nodes 5 and 1, respectively gives the optimum solution.

Using distances on the example of Fig. 3

Table 3 illustrates the use of distances to identify far-median candidates and compute the corresponding costs. The column labeled m' gives the far-median candidate and the cost column gives $\mathrm{cost}(\mathrm{T}[j], \mathrm{m}(j)) + \mathrm{cost}(T \setminus \mathrm{T}[j], m')$, which may differ from $\mathrm{cost}[j]$, the cost using the correct far median. In fact, there are only three cut edges where one of the far median candidates matches the correct far median (and $\mathrm{cost}[j]$ is computed correctly). One of these is the optimum solution.

Using binary search on the example of Fig. 3

Fig. 22 shows the binary search for $\mathrm{m}'(6)$. Cost computations are identical to those of the algorithm that sorts by weights.

B Another example with unit-length edges: a simple path

The example in Fig 23 is a simple path. Here the one-median m = node 0 and the cost $\mathrm{cost}(T, m) = 211$. There are two subtrees of equal weight with respect to m and two optimal solutions, shown in Fig. 24.

Tables 4 and 5 show the near median and associated cost computations for $\hat{T} = T_1$ and $\hat{T} = (T \setminus T_1) \setminus m$, respectively. On the T_1 side the near median moves only once, from node 4 to node 3.

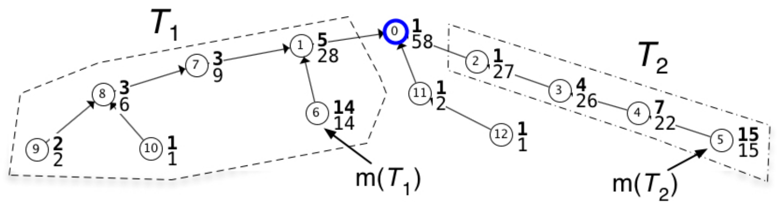


Fig. 14. Fig. 3 redrawn.

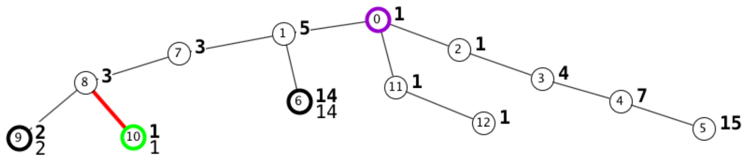


Fig. 15. Cut edge (10,8), m(T[10]) is 10, and m′(10) = 0.

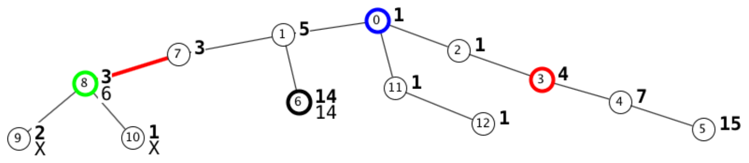


Fig. 16. Cut edge (8,7), m(T[8]) is 8, and m′(8) = 3;

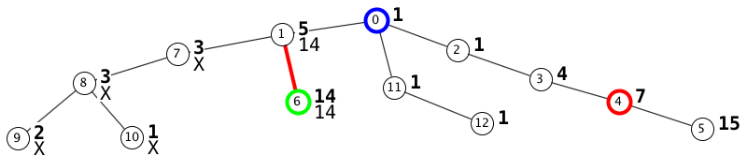


Fig. 17. Node 6 causes the far median to move again, to node 4 – the weight of $T\setminus$T[6] is 44 and w(T[4]) = 22, just half.

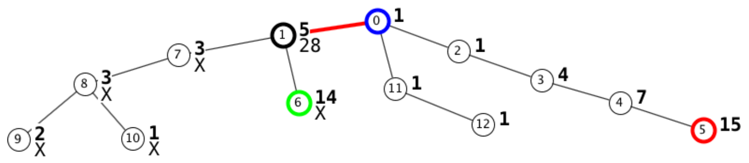


Fig. 18. Node 1 is removed with cut edge (1,0); w(T) − w(T[1]) = 30 and w(T[5]) is half that, so the far median moves to node 5.

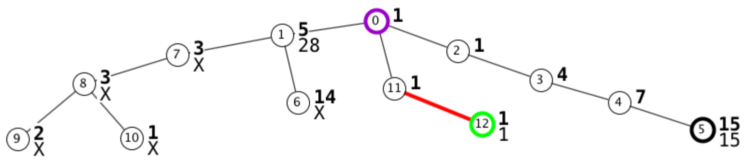


Fig. 19. Node 12 is considered with cut edge (12,11); w(T) − w(T[12]) = 57 and w(T[1]) is less than half that, so the far median stays at m.

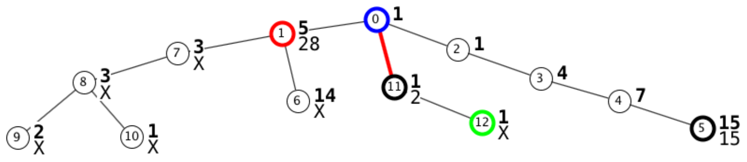


Fig. 20. Node 11 and cut edge $(11, m)$ are considered; $\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[11]) = 54$ and $\mathrm{w}(\mathrm{T}[1])$ is 28, more than half that, but $\mathrm{w}(\mathrm{T}[6]) = 14$, so the far median is node 1.

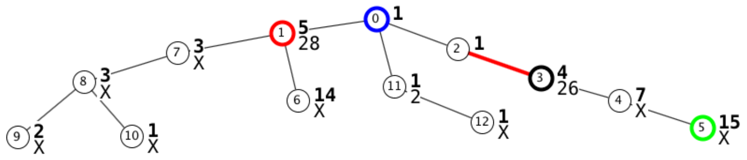
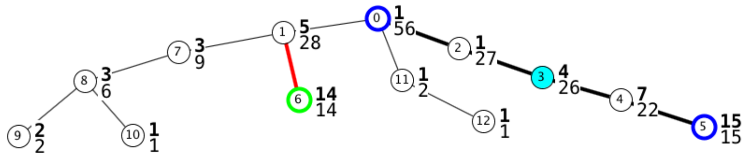
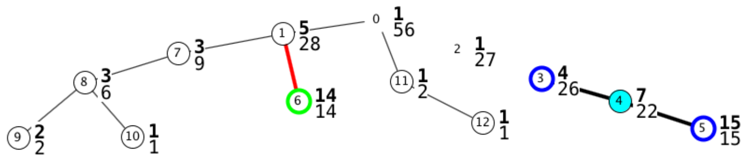


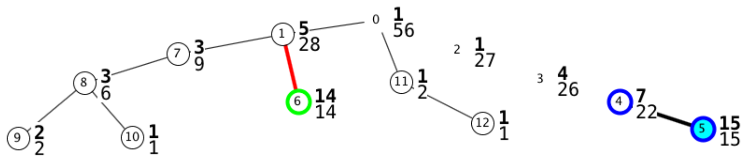
Fig. 21. The cut edge $(3, 2)$ with minimum cost. Near median is node 5, far median is node 1 and $\mathrm{cost}[3] = 55$.



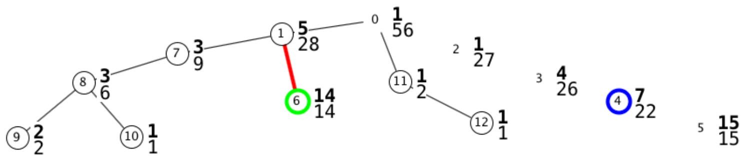
(a) Seaching path $[0, 5]$, $mid = 3$, $\mathrm{w}(\mathrm{T}[mid]) = 26 \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2 = 21$.



(b) Seaching path $[3, 5]$, $mid = 4$, $\mathrm{w}(\mathrm{T}[mid]) = 22 \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$.



(c) Seaching path $[4, 5]$, $mid = 5$, $\mathrm{w}(\mathrm{T}[mid]) = 15 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$.



(c) Seaching path $[4, 5) = [4, 4]$; only one node left, so $\mathrm{m}(T \setminus \mathrm{T}[j])$ is node 4.

Fig. 22. The example of Fig. 3, illustrating the binary search when j is node 6.

Table 3. Using distances to find far-median candidates in the example of Fig. 3. Correct far medians are marked with *****.

far median candidates for $\hat{T} = T_1$							
j	p_j	$\mathrm{m}(j)$	d_i	m'	$\mathrm{cost_m}[j]$	$\Delta(m')$	cost
10	8	10	$d_1 = 1 - 3 = -2$	ignore			
9	8	9	$d_1 = 1 - 3 = -2$	ignore			
8	7	8	$d_1 = 1 - 2 = -1$	ignore			
7	1	8	$d_1 = 2 - 1 = 1$	2	132	4	$132 + 4 - 1 \cdot \mathrm{w(T[7])} = 127$
			$d_2 = 1 - 1 = 0$	0	–	0	132
			$d_3 = 1 - 2 = -1$	ignore			
6	1	6	$d_1 = 1 - 1 = 0$	0	125	0	125
			$d_2 = 0 - 1 = -1$	ignore			
1	0	6	$d_1 = 2 - 0 = 2$	3	125	10	$125 + 10 - 2 \cdot \mathrm{w(T[1])} = 79$
			$d_2 = 1 - 0 = 1$	2	–	4	$125 + 4 - 1 \cdot \mathrm{w(T[1])} = 101$
			$d_3 = 0 - 0 = 0$	0	–	0	125
far median candidates for $\hat{T} = (T \setminus T_1) \setminus m$							
j	p_j	$\mathrm{m}(j)$	d_i	m'	$\mathrm{cost_m}[j]$	$\Delta(m')$	cost
12	11	12	$d_1 = 1 - 1 = 0$	**0***	151	0	151
			$d_2 = 0 - 1 = -1$	ignore			
11	0	12	$d_1 = 2 - 0 = 2$	6	151	32	$151 + 32 - 2 \cdot \mathrm{w(T[11])} = 179$
			$d_2 = 1 - 0 = 1$	**1***	–	10	$151 + 2 - 1 \cdot \mathrm{w(T[11])} = 151$
			$d_3 = 0 - 0 = 0$	0	–	0	151
5	4	5	$d_1 = 1 - 3 = -2$	ignore			
4	3	5	$d_1 = 2 - 2 = 0$	0	79	0	79
			$d_2 = 1 - 2 = -1$	ignore			
3	2	5	$d_1 = 3 - 1 = 2$	6	79	32	$79 + 32 - 2 \cdot \mathrm{w(T[3])} = 59$
			$d_2 = 2 - 1 = 2$	**1***	–	2	$79 + 2 - 1 \cdot \mathrm{w(T[3])} = \mathbf{55}$
			$d_3 = 2 - 2 = 0$	0	–	0	79
2	0	5	$d_1 = 4 - 0 = 4$	ignore (out of range)			
			$d_2 = 3 - 0 = 3$	ignore (out of range)			
			$d_3 = 3 - 1 = 2$	6	81	32	$81 + 32 - 2 \cdot \mathrm{w(T[2])} = 59$

This is because $\mathrm{w(T[4])} = 13 > \mathrm{w}(T_1)/2 = 12$. The near median does not move at all on the other side: $\mathrm{w(T[12])} = 15$ and $\mathrm{w}(T_2) = 24$.

The Δ values are as follows. Edge distance multipliers are omitted since all distances are 1. Since the median of T_1 is node 3 we consider nodes on the path $[1, 3]$:

$$\Delta(1) = \Delta(0) + (\mathrm{w}(T) - 2\mathrm{w(T[1])}) = 0 + 51 - 2 \cdot 24 = 3$$
$$\Delta(3) = \Delta(1) + (\mathrm{w}(T) - 2\mathrm{w(T[3])}) = 3 + 51 - 2 \cdot 13 = 28$$

On the T_2 side the median is node 12, so we must compute Δ for every node on the path $[2, 12]$

$$\Delta(2) = \Delta(0) + (\mathrm{w}(T) - 2\mathrm{w(T[2])}) = 0 + 51 - 2 \cdot 24 = 3$$
$$\Delta(5) = \Delta(2) + (\mathrm{w}(T) - 2\mathrm{w(T[5])}) = 3 + 51 - 2 \cdot 23 = 8$$

Table 4. Calculation of $\mathrm{cost_m}[j]$ for $j \in T_1$ in the example of Fig. 23.

j	p_j	k^*	$\mathrm{w(T}[j])$	$\mathrm{d[m}(k^*), j] - \mathrm{d}[m, j]$	$\mathrm{w(T}[j]) - \mathrm{w(T}[k^*])$	$\mathrm{cost_m}[j]$
4	3	N/A	1	$\mathrm{d}[j, m] = 3$		$211 - 3 \cdot 1 = 208$
3	1	4	13	$1 - 2 = -1$	$13 - 1 = 12$	$-12 \ldots$
. . . (median moves to 3)				$\mathrm{d}[k^*, j] \cdot (2\mathrm{w(T}[k^*]) - \mathrm{w(T}[j])) = 1 \cdot (2 - 13) = -11$		$208 - 12 - 11 = 185$
1	0	3	24	$1 - 1 = 0$	$24 - 13 = 11$	$185 - 0 \cdot 11 = 185$

Table 5. Calculation of $\mathrm{cost_m}[j]$ for $j \in (T \setminus T_1) \setminus m$ in the example of Fig. 23.

j	p_j	k^*	$\mathrm{w(T}[j])$	$\mathrm{d[m}(k^*), j] - \mathrm{d}[m, j]$	$\mathrm{w(T}[j]) - \mathrm{w(T}[k^*])$	$\mathrm{cost_m}[j]$
12	11	N/A	15	$\mathrm{d}[j, m] = 9$		$211 - 9 \cdot 15 = 76$
11	10	12	16	$1 - 8 = -7$	$16 - 15 = 1$	$76 - 7 \cdot 1 = 69$
10	9	11	17	$2 - 7 = -5$	$17 - 16 = 1$	$69 - 5 \cdot 1 = 64$
9	8	10	18	$3 - 6 = -3$	$18 - 17 = 1$	$64 - 3 \cdot 1 = 61$
8	7	9	19	$4 - 5 = -1$	$18 - 17 = 1$	$61 - 1 \cdot 1 = 60$
7	6	8	20	$5 - 4 = 1$	$20 - 19 = 1$	$60 + 1 \cdot 1 = 61$
6	5	7	21	$6 - 3 = 3$	$21 - 20 = 1$	$61 + 3 \cdot 1 = 64$
5	2	6	23	$7 - 2 = 5$	$23 - 21 = 2$	$64 + 5 \cdot 2 = 74$
2	0	5	24	$8 - 1 = 7$	$24 - 23 = 1$	$74 + 7 \cdot 1 = 81$

$$\begin{aligned}
\Delta(6) &= \Delta(5) + (\mathrm{w}(T) - 2\mathrm{w(T}[6])) = 8 + 51 - 2 \cdot 21 = 17\\
\Delta(7) &= \Delta(6) + (\mathrm{w}(T) - 2\mathrm{w(T}[7])) = 17 + 51 - 2 \cdot 20 = 28\\
\Delta(8) &= \Delta(7) + (\mathrm{w}(T) - 2\mathrm{w(T}[8])) = 28 + 51 - 2 \cdot 19 = 41\\
\Delta(9) &= \Delta(8) + (\mathrm{w}(T) - 2\mathrm{w(T}[9])) = 41 + 51 - 2 \cdot 18 = 56\\
\Delta(10) &= \Delta(9) + (\mathrm{w}(T) - 2\mathrm{w(T}[10])) = 56 + 51 - 2 \cdot 17 = 63\\
\Delta(11) &= \Delta(10) + (\mathrm{w}(T) - 2\mathrm{w(T}[11])) = 63 + 51 - 2 \cdot 16 = 82\\
\Delta(12) &= \Delta(11) + (\mathrm{w}(T) - 2\mathrm{w(T}[12])) = 82 + 51 - 2 \cdot 15 = 103
\end{aligned}$$

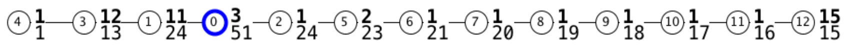


Fig. 23. The tree T, m is node 0, T_1 could be either the subtree rooted at node 1, i.e., the path [4,1], or the one rooted at node 2, the path [2,12]. The former is chosen arbitrarily so T_2 is the latter. The algorithms are run with $\hat{T} = T_1$ and again with $\hat{T} = T_2$.

Binary search on the example of Fig 23

This example was designed primarily to illustrate the actions of the binary search algorithm.

Fig. 25(b)–(e) shows the binary search for $\mathrm{m}'(j)$ when j is node 4. Since the far median is m, $\mathrm{cost}[4] = \mathrm{cost_m}[4] = 208$.

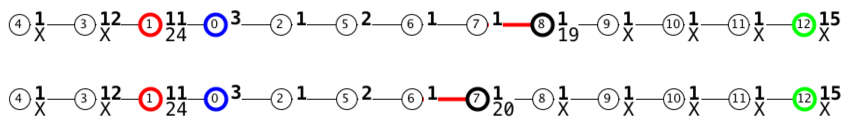
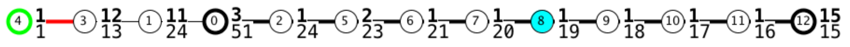
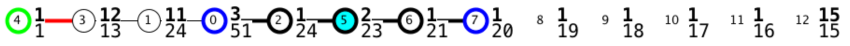


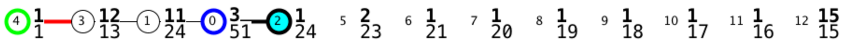
Fig. 24. Two optimal solutions for the example in Fig. 23.



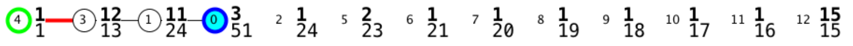
(a) Start with j = node 4. Searching path $[0, 12]$, $mid = 8$, $\mathrm{w}(\mathrm{T}[8]) = 19 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2 = 50/2 = 25$.



(b) Searching path $[0, 8) = [0, 7]$, $mid = 5$, $\mathrm{w}(\mathrm{T}[5]) = 23 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$.

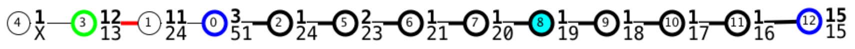


(c) Searching path $[0, 5) = [0, 4]$, $mid = 2$, $\mathrm{w}(\mathrm{T}[2]) = 24 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$.

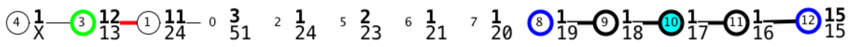


(d) Searching path $[0, 2) = [0, 0]$; only one node left, so $\mathrm{m}(T \setminus \mathrm{T}[j])$ is node 0.

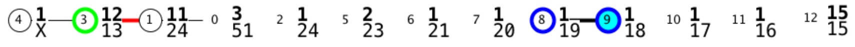
Fig. 25. Binary search for the far median of node 4, cut edge (4,3), in the tree of Fig. 23.



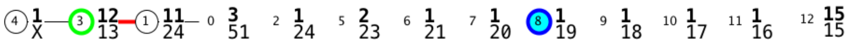
(a) Start with j = node 3. Searching path $[0, 12]$, $mid = 8$,
$\mathrm{w}(\mathrm{T}[8]) = 19 \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2 = (51 - 13)/2 = 19$.



(c) Searching path $[8, 12]$, $mid = 10$, $\mathrm{w}(\mathrm{T}[10]) = 17 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$.



(d) Searching path $[8, 10) = [8, 9]$, $mid = 9$, $\mathrm{w}(\mathrm{T}[9]) = 18 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[j]))/2$.



(e) Searching path $[8, 9) = [8, 8]$; only one node left, so $\mathrm{m}(T \setminus \mathrm{T}[j])$ is node 8.

Fig. 26. Binary search for the far median of node 3, cut edge (3,1), in the tree of Fig. 23.

Fig. 26 shows the search when j is node 3. Now the far median is node 8, so

$$\mathrm{cost}[3] = \mathrm{cost_m}[3] + \mathit{\Delta}(8) - \mathrm{d}[8, m] \cdot \mathrm{w}(\mathrm{T}[3]) = 185 + 41 - 5 \cdot 13 = 161$$

The search for m(1), the next j to be considered, is not shown. We are looking for the last m^* on the path $[0, 12]$ such that $\mathrm{w}(\mathrm{T}[m^*]) \geq (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[1]))/2 = (51 - 24)/2 = 13.5$. The search starts like the others, with $\mathit{mid} = 8$, then continues with path $[8, 12]$ and $\mathit{mid} = 10$; path $[10, 12]$ with $\mathit{mid} = 11$; and path $[11, 12]$ with $\mathit{mid} = 12$; since $\mathrm{w}(\mathrm{T}[12]) = 15 \geq 13.5$, the search is reduced to a single node, node 12.

$$\mathrm{cost}[1] = \mathrm{cost_m}[1] + \mathit{\Delta}(12) - \mathrm{d}[12, m] \cdot \mathrm{w}(\mathrm{T}[1]) = 185 + 103 - 9 \cdot 24 = 82$$

Steps of the algorithm with $\hat{T} = T_2$ are summarized as follows. The far median is always at node 1 because $\mathrm{w}(\mathrm{T}[1]) = 24 > (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[12]))/2 = 18$ and $\mathrm{w}(\mathrm{T}[3]) = 13 < (\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[2]))/2) = 13.5$.

$$\begin{aligned}
\mathrm{cost}[12] &= \mathrm{cost_m}[12] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[12]) = 76 + 3 - 15 = 64 \\
\mathrm{cost}[11] &= \mathrm{cost_m}[11] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[11]) = 69 + 3 - 16 = 56 \\
\mathrm{cost}[10] &= \mathrm{cost_m}[10] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[10]) = 64 + 3 - 17 = 50 \\
\mathrm{cost}[9] &= \mathrm{cost_m}[9] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[9]) = 61 + 3 - 18 = 46 \\
\mathrm{cost}[8] &= \mathrm{cost_m}[8] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[8]) = 60 + 3 - 19 = 44 \quad \text{(optimum)} \\
\mathrm{cost}[7] &= \mathrm{cost_m}[7] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[7]) = 61 + 3 - 20 = 44 \quad \text{(also optimum)} \\
\mathrm{cost}[6] &= \mathrm{cost_m}[6] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[6]) = 64 + 3 - 21 = 46 \\
\mathrm{cost}[5] &= \mathrm{cost_m}[5] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[5]) = 74 + 3 - 23 = 54 \\
\mathrm{cost}[2] &= \mathrm{cost_m}[2] + \mathit{\Delta}(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[2]) = 81 + 3 - 24 = 60
\end{aligned}$$

Sorting by weights on the example of Fig 23

The algorithm based on sorting by weights – Section 5 – does not need to sort. If a priority queue is used it never contains more than one element. The procedure is trivial and follows the near median algorithm along with the cost calculations shown above.

Distance based algorithm on the example of Fig 23

The computations of the distance-based algorithm are shown in Table 6. The first thing to notice is that, for $j \in T_1$, none of the correct far medians are identified and the costs are much larger than the correct ones. This does not matter, of course, because the two optimum solutions involve j and corresponding cut edges in T_2. On the T_2 side, the only (two) correctly identified far medians lead to optimum solutions.

Table 6. Using distances to find far-median candidates for the example in Fig. 23. Correct far medians are marked with *****.

far median candidates for $\hat{T} = T_1$							
j	p_j	$\mathrm{m}(j)$	d_i	m'	$\mathrm{cost_m}[j]$	$\Delta(m')$	cost
4	3	4	$d_1 = 1 - 2 = -1$	ignore			
3	1	3	$d_1 = 1 - 1 = 0$	0	185	0	185
			$d_2 = 0 - 1 = -1$	ignore			
1	0	3	$d_1 = 2 - 0 = 2$	5	185	8	$185 + 8 - 2 \cdot 24 = 145$
			$d_2 = 1 - 0 = 1$	2	–	3	$185 + 3 - 1 \cdot 24 = 164$
			$d_3 = 1 - 1 = 0$	0	–	0	185
far median candidates for $\hat{T} = (T \setminus T_1) \setminus m = T_2$							
j	p_j	$\mathrm{m}(j)$	d_i	m'	$\mathrm{cost_m}[j]$	$\Delta(m')$	cost
12	11	12	$d_1 = 1 - 8 = -7$	ignore			
11	10	12	$d_1 = 2 - 7 = -5$	ignore			
10	9	12	$d_1 = 3 - 6 = -3$	ignore			
9	8	12	$d_1 = 4 - 5 = -1$	ignore			
8	7	12	$d_1 = 5 - 4 = 1$	**1***	60	3	$60 + 3 - 1 \cdot \mathrm{w(T[8])} = \mathbf{44}$
			$d_2 = 4 - 4 = 0$	0	–	0	60
			$d_3 = 4 - 5 = -1$	ignore			
7	6	12	$d_1 = 6 - 3 = 3$	ignore: node 4 $\notin [0, 1]$			
			$d_2 = 5 - 3 = 2$	ignore: node 3 $\notin [0, 1]$			
			$d_3 = 5 - 4 = 1$	**1***	61	3	$61 + 3 - 1 \cdot \mathrm{w(T[7])} = \mathbf{44}$
6	5	12	$d_3 = 6 - 3 = 3$	ignore: node 4 $\notin [0, 1]$			
			$d_1 > d_2 > d_3$ so no point considering				
5	2	12	$d_3 = 7 - 2 = 5$	ignore: all d_i out of range			
2	0	12	$d_3 = 8 - 1 = 7$	ignore: all d_i out of range			

C An Example with Non-Unit Edge Lengths

The example shown in Fig. 27 illustrates the more complex calculations involved when edge lengths are not $= 1$. It also illustrates a situation where the near median jumps more than one position, i.e., $\mathrm{m}(j)$ is more than one edge away from $\mathrm{m}(k^*)$, resulting in more than one iteration of line 8 in the algorithm of Fig. 9. See Fig. 28.

The one-median m is node 0, T_1 and T_2 are rooted at nodes 1 and 2, respectively, and have medians at nodes 7 and 2. The cost of the whole tree with respect to the one-median, $\mathrm{cost}(T, m) = 264$.

Tables 7 and 8 show the steps taken by the near median algorithm, Fig. 9, on the example.

Leaf nodes 3, 4, and 5 are processed first. Here we use the (non-unit) edge lengths to compute $\mathrm{cost_m}[j] = \mathrm{d}[j, m] \cdot \mathrm{w(T}[j])$.

The near median stays at node 5 when node 6 is considered: $\mathrm{w(T[5])} = 5 \geq \mathrm{w(T[6])}/2 = 4$.

However, as shown in Fig. 28, it moves to node 7 when that node is processed: $\mathrm{w(T[7])}/2 = 8.5$. This is greater than $\mathrm{w(T[6])} = 8$ and certainly greater than $\mathrm{w(T[5])}$. The cost updates in the two iterations of the loop at line 8 are shown in the table.

The near median $\mathrm{m}(1)$ is still at node 7: $\mathrm{w(T[7])} = 17 \geq \mathrm{w(T[1])}/2 = 11$.

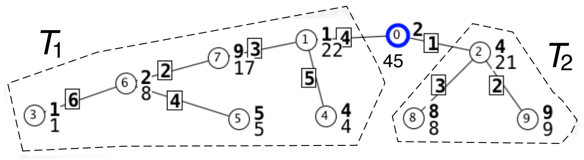


Fig. 27. A tree with non-unit edge lengths. The edge lengths are shown in the boxes near the midpoints of the edges.

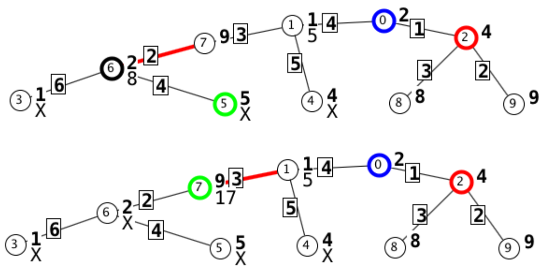


Fig. 28. A jump from m(6) = 5 to m(7) = 7.

Table 7. Calculation of $\text{cost}_\text{m}[j]$ for $j \in T_1$ in the example of Fig. 27.

j	p_j	k^*	$\text{w}(\text{T}[j])$	$\text{d}[\text{m}(k^*), j] - \text{d}[m, j]$	$\text{w}(\text{T}[j]) - \text{w}(\text{T}[k^*])$	$\text{cost}_\text{m}[j]$
3	6	N/A	1	$\text{d}[j, m] = 6 + 2 + 3 + 4 = 15$		$264 - 15 \cdot 1 = 249$
4	1	N/A	4	$\text{d}[j, m] = 5 + 4 = 9$		$264 - 9 \cdot 4 = 228$
5	6	N/A	5	$\text{d}[j, m] = 4 + 2 + 3 + 4 = 13$		$264 - 13 \cdot 5 = 199$
6	7	5	8	$4 - 9 = -5$	$8 - 5 = 3$	$199 - 5 \cdot 3 = 184$
7	1	6[a]	17	$6 - 7 = -1$	$17 - 8 = 9$	$184 - 9 = 175 \ldots$
$\ldots x = 5,\ \text{p}_x = 6$				$\text{d}[x, \text{p}_x] \cdot (2\text{w}(\text{T}[x]) - \text{w}(\text{T}[j])) = 4 \cdot (2 \cdot 5 - 17) = -28$		$\ldots - 28 = 147 \ldots$
$\ldots x = 6,\ \text{p}_x = 7$				$\text{d}[x, \text{p}_x] \cdot (2\text{w}(\text{T}[x]) - \text{w}(\text{T}[j])) = 2 \cdot (2 \cdot 8 - 17) = -2$		$\ldots - 2 = 145$
1	0	7	22	$3 - 4 = -1$	$22 - 17 = 5$	$145 - 1 \cdot 5 = 140$

[a] m(k^*) is node 5.

Table 8. Calculation of $\text{cost}_\text{m}[j]$ for $j \in (T \setminus T_1) \setminus m = T_2$ in the example of Fig. 27.

j	p_j	k^*	$\text{w}(\text{T}[j])$	$\text{d}[\text{m}(k^*), j] - \text{d}[m, j]$	$\text{w}(\text{T}[j]) - \text{w}(\text{T}[k^*])$	$\text{cost}_\text{m}[j]$
8	2	N/A	8	$\text{d}[j, m] = 3 + 1 = 15$		$264 - 4 \cdot 8 = 232$
9	2	N/A	9	$\text{d}[j, m] = 2 + 1 = 3$		$264 - 3 \cdot 9 = 237$
2	0	9	21	$2 - 1 = 1$	$21 - 9 = 12$	$237 + 12 = 249 \ldots$
$\ldots x = 9,\ \text{p}_x = 2$				$\text{d}[x, \text{p}_x] \cdot (2\text{w}(\text{T}[x]) - \text{w}(\text{T}[j])) = 2 \cdot (2 \cdot 9 - 21) = -6$		$\ldots - 6 = 243$

On the T_2 side, the leaves are nodes 8 and 9. The only other node, node 2, is its own near median and the relevant cost updates are as indicated.

Delta values are as follows. First in T_1, along the path $[1, 7]$.

$$\begin{aligned}\Delta(1) &= \Delta(0) + d_{1,0} \cdot (\mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[1])) = 0 + 4 \cdot (45 - 2 \cdot 22) = 4\\ \Delta(7) &= \Delta(1) + d_{7,1} \cdot (\mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[7])) = 4 + 3 \cdot (45 - 2 \cdot 17) = 37\end{aligned}$$

Then the T_2, along the path $[2, 2]$.

$$\Delta(2) = \Delta(0) + d_{2,0} \cdot (\mathrm{w}(T) - 2\mathrm{w}(\mathrm{T}[2])) = 0 + 1 \cdot (45 - 2 \cdot 21) = 3$$

We omit the trace of the binary search algorithm on this example: the lengths of the paths being searched are two on the T_1 side and only one on the T_2 side. This means that most of the work is done by the near median algorithm. The trace of sorting by weights focuses mainly on cost computations, while that of the distance-based algorithm focuses on the limited number of far median *candidates* under consideration.

Sorting by weights on the example of Fig 27

The nodes of T_1, in increasing subtree weight order, are: 3, 4, 5, 6, 7, 1; they are listed in that same order in Table 7. Far medians progress as follows:

node 3: $\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[3]) = 44$; $\mathrm{w}(\mathrm{T}[2]) = 21 < 44/2$ $\mathrm{m}'(3) = m = 0$
node 4: $\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[4]) = 41$; $\mathrm{w}(\mathrm{T}[2]) = 21 \geq 41/2$ $\mathrm{m}'(4) =$ node 2

The remaining far medians are at node 2; that is as far as they can travel because $\mathrm{m}(T_2) =$ node 2.

Nodes of T_2 also appear in their increasing subtree weight order in Table 8: 8, 9, 2.

Far medians are as follows:

node 8: $\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[8]) = 37$; $\mathrm{w}(\mathrm{T}[1]) = 22 > 37/2$, $\mathrm{w}(\mathrm{T}[7]) = 17 < 37/2$ $\mathrm{m}'(8) =$ node 1
node 9: $\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[9]) = 36$; $\mathrm{w}(\mathrm{T}[7]) = 17 < 37/2$ $\mathrm{m}'(9)$ stays at node 1
node 2: $\mathrm{w}(T) - \mathrm{w}(\mathrm{T}[2]) = 24$; $\mathrm{w}(\mathrm{T}[7]) = 17 \geq 24/2$ and node 7 is $\mathrm{m}(T_1)$ $\mathrm{m}'(2) =$ node 7

Cost computation based on Δ values are straightforward. On the T_1 side:

$$\begin{aligned}
\mathrm{cost}[3] &= \mathrm{cost_m}[3] \text{ (no adjustment, } \mathrm{m}'(3) \text{ is } m) = && && 249\\
\mathrm{cost}[4] &= \mathrm{cost_m}[4] + \Delta(2) - \mathrm{d}[2, m] \cdot \mathrm{w}(\mathrm{T}[4]) &&= 228 + 3 - 1 \cdot 4 &&= 227\\
\mathrm{cost}[5] &= \mathrm{cost_m}[5] + \Delta(2) - \mathrm{d}[2, m] \cdot \mathrm{w}(\mathrm{T}[5]) &&= 199 + 3 - 1 \cdot 5 &&= 197\\
\mathrm{cost}[6] &= \mathrm{cost_m}[6] + \Delta(2) - \mathrm{d}[2, m] \cdot \mathrm{w}(\mathrm{T}[6]) &&= 184 + 3 - 1 \cdot 8 &&= 179\\
\mathrm{cost}[7] &= \mathrm{cost_m}[7] + \Delta(2) - \mathrm{d}[2, m] \cdot \mathrm{w}(\mathrm{T}[7]) &&= 145 + 3 - 1 \cdot 17 &&= 131\\
\mathrm{cost}[1] &= \mathrm{cost_m}[1] + \Delta(2) - \mathrm{d}[2, m] \cdot \mathrm{w}(\mathrm{T}[1]) &&= 140 + 3 - 1 \cdot 22 &&= \mathbf{121}
\end{aligned}$$

And on the T_2 side:

$$\begin{aligned}
\mathrm{cost}[3] &= \mathrm{cost_m}[3] \text{ (no adjustment, } \mathrm{m}'(3) \text{ is } m) = && && 249\\
\mathrm{cost}[8] &= \mathrm{cost_m}[8] + \Delta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[8]) &&= 232 + 4 - 4 \cdot 8 &&= 204\\
\mathrm{cost}[9] &= \mathrm{cost_m}[9] + \Delta(1) - \mathrm{d}[1, m] \cdot \mathrm{w}(\mathrm{T}[9]) &&= 237 + 4 - 4 \cdot 9 &&= 205\\
\mathrm{cost}[2] &= \mathrm{cost_m}[2] + \Delta(7) - \mathrm{d}[7, m] \cdot \mathrm{w}(\mathrm{T}[2]) &&= 243 + 37 - 7 \cdot 21 &&= 133
\end{aligned}$$

Using distances on the tree of Fig. 27

The first step in the distance-based algorithm is to create, for each side, the array A that stores the nodes based on their distance from m. On the T_1 side, the non-**null** entries are

$$\begin{aligned} A[0] &= m = \text{node } 0 \\ A[4] &= \quad\;\; \text{node } 1 \\ A[7] &= \quad\;\; \text{node } 7 \end{aligned}$$

On the T_2 side they are

$$\begin{aligned} A[0] &= m = \text{node } 0 \\ A[1] &= \quad\;\; \text{node } 2 \end{aligned}$$

Most of the d values for each j will be out of range. In what follows, we show only the ones that range from 0 to 1 when we process T_1 and from 0 to 7 when we process T_2. The trace is shown in Table 9. In other cases we give the *delta* value that puts d out of range, with the implication that all smaller/larger values will be out of range as well. The **null** values for $A[d]$ when $j \in T_2$ reflect the fact that the location of the other median, in order for the midpoint to as advertised, would be somewhere along the edge (1,0) rather than either of its endpoints.

Table 9. Using distances to find far-median candidates for the example in Fig. 27. Correct far medians are marked with *****.

far median candidates for $\hat{T} = T_1$								
j	p_j	$\mathrm{m}(j)$	$\mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m]$	δ	d	$A[d]$	$\mathrm{cost}_\mathrm{m}[j]$	cost
3	6	3	$0 - 15 = -15$	$-15 + 2 \cdot 6 < 0$	ignore			
4	1	4	$0 - 9 = -9$	δ ranges from 4.5 to 5				
				4.5	$-9 + 2 \cdot 4.5 = 0$	0	228	228
				5	$-9 + 2 \cdot 5 = 1$	**2***	228	227
5	6	5	$0 - 13 = -13$	$0 - 13 + 2 \cdot 4 < 0$	ignore			
6	7	5	$4 - 9 = -5$	δ ranges from 2.5 to 3				
				2.5	$-5 + 2 \cdot 2.5 = 0$	0	184	184
				3	$-5 + 2 \cdot 3 = 1$	**2***	184	161
7	1	7	$0 - 7 = -7$	$-7 + 2 \cdot 3 < 0$	ignore			
1	0	7	$3 - 4 = -1$	δ ranges from 0.5 to 1				
				0.5	$-1 + 2 \cdot 0.5 = 0$	0	140	140
				1	$-1 + 2 \cdot 1 = 1$	**2***	140	**121**
far median candidates for $\hat{T} = (T \setminus T_1) \setminus m = T_2$								
j	p_j	$\mathrm{m}(j)$	$\mathrm{d}[\mathrm{m}(j), j] - \mathrm{d}[j, m]$	δ	d	$A[d]$	$\mathrm{cost}_\mathrm{m}[j]$	cost
8	2	8	$0 - 4 = -4$	δ ranges from 2 to 3				
				2	$-4 + 2 \cdot 2 = 0$	0	232	232
				2.5	$-4 + 2 \cdot 2.5 = 1$	**null**		
				3	$-4 + 2 \cdot 3 = 2$	**null**		
9	2	9	$0 - 3 = -3$	δ ranges from 1.5 to 2				
				1.5	$-3 + 2 \cdot 1.5 = 0$	0	237	237
				2	$-3 + 2 \cdot 2 = 1$	**null**		
2	0	2	$0 - 1 = -1$	δ ranges from 0.5 to 1				
				0.5	$-1 + 2 \cdot 0.5 = 0$	0	243	243
				2	$-1 + 2 \cdot 1 = 1$	**null**		