Algorithmica (2014) 70:2–21
DOI 10.1007/s00453-013-9851-7

A Linear Time Algorithm for Computing Minmax Regret 1-Median on a Tree Network

Binay Bhattacharya · Tsunehiko Kameda · Zhao Song

Received: 3 October 2012 / Accepted: 5 November 2013 / Published online: 16 November 2013

Abstract In a model of facility location problem, the uncertainty in the weight of a vertex is represented by an interval of weights, and the objective is to minimize the maximum "regret." The most efficient algorithm previously known for finding the minmax regret 1-median in a tree network with nonnegative vertex weights takes $O(n \log n)$ time. We improve it to $O(n)$, settling the open problem posed by Brodal et al. (Oper. Res. Lett. 36:14–18, 2008).

Keywords Facility location · Robust median · Uncertain weights · Minmax regret

1 Introduction

1.1 The Problem

Deciding where to locate facilities to minimize the communication or transportation costs is known as the *facility location problem*. For a recent review of this subject, the reader is referred to [10]. The cost function is formulated as the sum of the distances from the nearest facility weighted by the weights of the vertices. In the *minmax regret* version of this problem, there is uncertainty in the weights of the vertices and/or edge

Zhao Song took part in this work while he was an undergraduate student at SFU.

B. Bhattacharya · T. Kameda (✉)
School of Computing Science, Simon Fraser University, Burnaby, Canada
e-mail: tiko@sfu.ca

B. Bhattacharya
e-mail: binay@sfu.ca

Z. Song
Department of Computer Science, The University of Texas at Austin, Austin, TX, USA
e-mail: zhaos@utexas.edu

lengths, and only their ranges are known [8, 12, 13]. Chen and Lin (Theorem 1 in [8]) proved that in solving this problem, the edge lengths can be set to their maximum values. Therefore, we assume that the (positive) edge lengths are fixed and uncertainty is only in the vertex weights. A particular *realization* (assignment of a weight to each vertex) is called a *scenario*. Intuitively, the minmax regret 1-median problem can be understood as a 2-person game as follows. The first player picks a location x to place a facility. The opponent's move is to pick a scenario s. The payoff to the second player is the cost of x minus the cost of the median, both under s, and he wants to pick the scenario s that maximizes his payoff. Our objective (as the first player) is to select x that minimizes this payoff in the worst case, i.e., over all scenarios.

1.2 Previous Work

The problem of finding the minmax regret 1-median on a network, and a tree network in particular, has been attracting great research interest in recent years, and many researchers have worked on this problem. Kouvelis et al. formulated the problem of finding the minmax regret 1-median on a tree and proposed an $O(n^4)$ solution, where n is the number of vertices [13]. Chen and Lin improved it to $O(n^3)$ [8]. Averbakh and Berman then found a simple $O(n^2)$ algorithm [1] and improved it later to $O(n \log^2 n)$ [2]. Yu et al. [16] proposed an $O(n \log n)$ implementation of the algorithm in [2]. More recently, Brodal et al. also came up with a simpler $O(n \log n)$ algorithm [5]. When the vertices can have negative weights, Burkard and Dollani gave an $O(n^2)$ algorithm [6]. Recently, we improved it to $O(n \log^2 n)$ [4]. This paper is a full version of our extended abstract presented at COCOON 2012 [3].

In this paper, we present an $O(n)$ time algorithm for trees when the edge lengths are fixed, and each vertex has a weight from an interval of nonnegative values. This settles the open problem posed in [5]. In their $O(n \log n)$-time algorithms, Yu et al. [16] and Brodal et al. [5] successively reduce the size of the part of the tree in which the optimal location lies. Even though these algorithms reduce this size geometrically, they need to spend $O(n)$ time in each round. The main contribution of this paper is to reduce this time to the order of the size of the remaining part of the tree in each round, which reduces the overall time requirement to $O(n)$. To achieve this, we resort to a few somewhat elaborate algorithms. Our pruning algorithm was inspired by a method used by Megiddo [14].

1.3 Paper Organization

The rest of this paper is organized as follows. In the next section, we first review definitions and some known facts, and then prove a lemma (Lemma 4), which restricts the set of scenarios we need to consider. Section 3 describes methods for computing the medians for the scenarios of interest, and for computing their costs. Section 4 is devoted to the detailed discussion of our main algorithm and its time complexity analysis. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 Definitions

Let $T = (V, E)$ be a tree network (or simply a tree) with n vertices. We also use T to denote the set of all points (vertices and points on edges) on T. Each vertex $v \in V$ is associated with an interval of nonnegative integer weights $W(v) = [\underline{w}_v, \overline{w}_v]$, where $0 \le \underline{w}_v \le \overline{w}_v$, and each edge $e \in E$ is associated with a positive length (or distance). In order to exclude a trivial case, we assume $\underline{w}_v > 0$ for at least one vertex. For any two points $a, b \in T$, $d(a, b)$ denotes the shortest distance between a and b on T. If a and/or b is on an edge, then the distance is a prorated fraction of its length. Let $\mathcal{S}$ denote the Cartesian product of all $W(v)$, $v \in V$:

$$\mathcal{S} \triangleq \prod_{v \in V} [\underline{w}_v, \overline{w}_v].$$

Under a scenario $s \in \mathcal{S}$, we define the *cost* of a point $x \in T$ by

$$F^s(x) \triangleq \sum_{v \in V} d(v, x) w_v^s, \tag{1}$$

where w_v^s denotes the weight of v under s. A location x that minimizes (1) is a 1-*median* under s. Throughout this paper we use 1-median and *median* synonymously. We call

$$R^s(x) \triangleq F^s(x) - F^s\big(m(s)\big) \tag{2}$$

the *regret* [13] of point x under s, where $m(s)$ denotes a median under s. A scenario s is said to *dominate* another scenario s' at x if $R^s(x) \ge R^{s'}(x)$ holds. We finally define the *maximum regret*, $R^*(x)$, of x as the regret of the scenario that dominates all others at x.

$$R^*(x) \triangleq \max_{s \in \mathcal{S}} R^s(x). \tag{3}$$

Note that $R^*(x)$ is the maximum payoff with respect to x that we mentioned in the Introduction. We seek location $x^* \in T$, called the *minmax regret median*, that minimizes $R^*(x)$. We sometimes refer to x^* as the *optimal location*. Let $s = \hat{s}(x)$ maximize (2) for a given $x \in T$. We call $\hat{s}(x)$ and $m(\hat{s}(x))$ a *worst case scenario* and a *worst case alternative* for x, respectively [1].

2.2 Properties of Median and Minmax Regret Median in a Tree

Let $v \in V$ be a vertex with degree d connected to vertices $v_1, v_2, \ldots, v_d$. If we remove v and all edges incident to it from T, then d subtrees, $T(v_1), T(v_2), \ldots, T(v_d)$ result. Let $W(T(v_i))$ denote the total weight of the vertices in $T(v_i)$. Vertex v is said to be a *weight centroid* or *w-centroid* [11] if

$$W\big(T(v_i)\big) \le W(T)/2, \tag{4}$$

holds for all $i = 1, 2, \ldots, d$, where $W(T)$ denotes the total weight of the vertices in T.

Lemma 1 [9] *If the vertex weights are non-negative, a vertex is a median if and only if it is a w-centroid, and there is always a vertex that is a median.*

By Lemma 1 we shall assume that a median is always at a vertex. However, the minmax regret median may not be at a vertex [13]. If there exists a vertex u that is a median under all the scenarios, then clearly u is the minmax regret location. In such a case, the problem instance is said to be *degenerate*.

Lemma 2 *In a general graph, we have $R^*(x^*) = 0$ if and only if the problem instance is degenerate.*

Proof The if part is obvious. So suppose that $R^*(x^*) = 0$. This implies that under any scenario s we have $R^s(x^*) = F^s(x^*) - F^s(m(s)) = 0$, and thus x^* is a median under s, i.e., the problem instance is degenerate. □

In the rest of the paper, we shall assume that a given problem instance is not degenerate.

Lemma 3 [8, Theorem 1(a)] *Given any point $x \in T$, there exists a worst-case scenario $\hat{s}(x)$ such that $w_v^{\hat{s}(x)} = \underline{w}_v$ for any vertex v satisfying $d(x, v) < d(m(\hat{s}(x)), v)$, and $w_v^{\hat{s}(x)} = \overline{w}_v$ for any vertex v satisfying $d(x, v) > d(m(\hat{s}(x)), v)$. If $d(x, v) = d(m(\hat{s}(x)), v)$, then $w_v^{\hat{s}(x)}$ may take any value in $[\underline{w}_v, \overline{w}_v]$.*

Let $e = (v, v') \in E$, and let $T(v)$ (resp. $T(v')$) denote the maximal subtree of T that does not contain e but contains v (resp. v'). Let $s \in \mathcal{S}$ be such that $w_u^s = \overline{w}_u$ for each vertex $u \in T(v)$, and $w_u^s = \underline{w}_u$ for each $u \in T(v')$. Such a scenario s is called a *bipartite* scenario, and v is the *front* of s, denoted by $f(s)$. Let $\mathcal{S}^* \subset \mathcal{S}$ be the set of all bipartite scenarios. Under a scenario $s \in \mathcal{S}^*$, we call the component, each vertex of which has the max (resp. min) weight, the *max-weighted component* (resp. *min-weighted component*) and denote it by $\max(s)$ (resp. $\min(s)$). By definition, under any scenario $s \in \mathcal{S}^*$ both $\max(s)$ and $\min(s)$ are nonempty. Clearly, each vertex v is the front of $d(v)$ scenarios in $\mathcal{S}^*$, where $d(v)$ denotes the degree of v. Note that scenario $\hat{s}(x)$ in Lemma 3 can be made bipartite. Therefore, we have [1]

$$\forall x \in T : R^*(x) = \max_{s \in \mathcal{S}^*} R^s(x). \tag{5}$$

Averbakh et al. [1] remark that (5) holds for any network (tree or not).

Let $\tilde{\mathcal{S}} \subset \mathcal{S}^*$ be the set of scenarios under which a median is in the max-weighted component. The following lemma follows directly from Lemma 3.

Lemma 4

(a)

$$\forall x \in T : R^*(x) = \max_{s \in \tilde{\mathcal{S}}} R^s(x). \tag{6}$$

(b) *For any point $x \in T$, there is a worst case scenario $\hat{s}(x) \in \tilde{\mathcal{S}}$ such that x is not in its max-weighted component.*

By Lemma 4(a), we only consider the scenarios in $\tilde{\mathcal{S}}$ in the rest of this paper. Lemma 4(b) will be utilized in our pruning algorithm in Sect. 4.3. A major obstacle, however, is the fact that computing $\{m(s) \mid s \in \tilde{\mathcal{S}}\}$ takes $\Omega(n \log n)$ time (Theorem 6.1 in [7]). Much effort will be made to overcome this problem by computing only the medians of *relevant* scenarios.

3 Medians and Their Costs

Equation (6) implies that, in order to find the optimal location, we need to compare the regrets under the scenarios in $\tilde{\mathcal{S}}$. Since regret is defined by (2), our first task is to compute the medians of different scenarios in $\tilde{\mathcal{S}}$, especially the dominating ones. Given a tree T, we pick an arbitrary vertex r_0 as its root. Let $T(v)$ denote the subtree of T rooted at v. The subtree of T defined by the vertices that do not belong to $T(v)$ is called the *complement* of $T(v)$ and is denoted by $T^c(v)$. For a non-root vertex v, we use $p(v)$ to denote the parent of v.

3.1 Computing Median $m(s)$ for all $s \in \tilde{\mathcal{S}}$ with $\max(s) = T(v)$

Goldman computes a median of a tree under a scenario in linear time [9]. Here, for all $v \in V$, we compute the medians under every scenario $s \in \tilde{\mathcal{S}}$ such that $\max(s) = T(v)$ in linear time. For convenience, we define two arrays for *subtree weights*, $\overline{W}_t[\cdot]$ and $\underline{W}_t[\cdot]$, and two arrays for *complement weights*, $\overline{W}_c[\cdot]$ and $\underline{W}_c[\cdot]$, as follows.

$$\overline{W}_t[v] = \sum_{u \in T(v) \cap V} \overline{w}_u, \qquad \underline{W}_t[v] = \sum_{u \in T(v) \cap V} \underline{w}_u,$$

$$\overline{W}_c[v] = \sum_{u \in T^c(v) \cap V} \overline{w}_u, \qquad \underline{W}_c[v] = \sum_{u \in T^c(v) \cap V} \underline{w}_u.$$

We can easily compute $\overline{W}_t[\cdot]$ and $\underline{W}_t[\cdot]$ in $O(n)$ time. Once they have been computed, to compute $\overline{W}_c[v]$, for example, we can use the relation $\overline{W}_c[v] = \overline{W}_t[r_0] - \overline{W}_t[v]$. For two points $a, b \in T$, let $\pi(a, b)$ denote the shortest path between a and b. Let $m(s)$ denote the median vertex under scenario s that is the farthest from the root of T. The following lemma is implied by a result in [9].

Lemma 5 *Given a scenario $s \in \tilde{\mathcal{S}}$, let vertex v be defined by $T(v) = \max(s)$. For each vertex u on $\pi(m(s), v)$, $\overline{W}_t[u]$ is at least half the total vertex weight under s.*

Proof Clearly, it suffices to prove the assertion for the case $u = m(s)$. Since $m(s)$ is a median under s, by Lemma 1 we have

$$\overline{W}_t[v] + \underline{W}_c[v] - \overline{W}_t\big[m(s)\big] \le \big(\overline{W}_t[v] + \underline{W}_c[v]\big)/2,$$

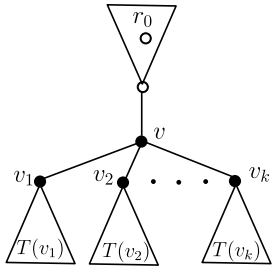

Fig. 1 $\max(s) = T(v)$ for scenario s

where $\overline{W}_t[v] + \underline{W}_c[v]$ is the total weight under s, and the left hand side is the weight of $T^c(m(s))$ under s. From the above inequality, it follows that

$$\overline{W}_t\big[m(s)\big] \geq \big(\overline{W}_t[v] + \underline{W}_c[v]\big)/2. \qquad \square$$

Let us now address the issue of computing $\{m(s) \mid s \in \tilde{\mathcal{S}} \text{ and } r_0 \notin \max(s)\}$ efficiently. We perform a post-order depth first traversal, carrying out the w-centroid test (4) on each vertex visited. When we visit vertex v during the traversal, we compute $m(s)$ for scenario s with $\max(s) = T(v)$, if $m(s) \in T(v)$ (i.e., $s \in \tilde{\mathcal{S}}$). See Fig. 1, where $v_1, v_2, \ldots, v_k$ are the child vertices of v. If $\underline{W}_c[v] > \overline{W}_t[v]$ then $m(s) \notin T(v)$ (i.e., $s \notin \tilde{\mathcal{S}}$) by Lemma 1. So assume $\underline{W}_c[v] \leq \overline{W}_t[v]$. For $j = 1, 2, \ldots, k$ let s_j be the scenario such that $\max(s_j) = T(v_j)$, and assume also that $m(s_j)$, such that $m(s_j) \in T(v_j)$ (i.e., $s_j \in \tilde{\mathcal{S}}$), has already been computed. If $m(s_j) \notin T(v_j)$ (i.e., $s_j \notin \tilde{\mathcal{S}}$) for all j, then we have $m(s) = v$, because $m(s_j) \notin T(v_j)$ implies $m(s) \notin T(v_j)$. Let us now assume that there is an index j with $m(s_j) \in T(v_j)$ (i.e., $s_j \in \tilde{\mathcal{S}}$). Lemma 5 implies that $m(s)$ lies either in subtree $T(v_j)$ with the largest $\overline{W}_t[v_j]$ or at v.

Lemma 6 *Let s be a scenario such that $\max(s) = T(v)$ for a vertex v that has v_j as a child vertex. If there is a median under s that lies in subtree $T(v_j)$, then there is a median under s that lies on $\pi(m(s_j), v_j)$.*

Proof Suppose that there is a median under s that lies in subtree $T(v_j)$. Then, by Lemma 5 we have

$$\overline{W}_t[v_j] \geq \big\{\overline{W}_t[v] + \underline{W}_c[v]\big\}/2. \tag{7}$$

Since the weight of any vertex in $T(v)$ under s is not smaller than that under s_j, $m(s_j)$ must also lie in $T(v_j)$. However, $m(s)$ cannot lie in $T(m(s_j))$, except possibly at $m(s_j)$. We clearly have

$$\forall u \in \pi\big(m(s_j), v_j\big) : \overline{W}_t[u] \geq \big\{\overline{W}_t[v_j] + \underline{W}_c[v_j]\big\}/2. \tag{8}$$

From (7) and (8), it follows that

$$\exists u \in \pi\big(m(s_j), v_j\big) : \overline{W}_t[u] \geq \big\{\overline{W}_t[v] + \underline{W}_c[v]\big\}/2, \tag{9}$$

and we have $m(s) = u$, where u $(\in \pi(m(s_j), v_j))$ is the vertex closest to $m(v_j)$ satisfying (9). $\square$

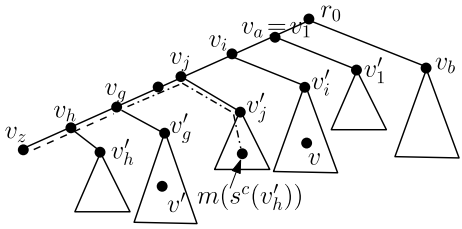

Fig. 2 Positions of $v_g = m_{z+1}$, v_h, and $m(s^c(v'_h))$

Theorem 1 *For all $v \in V$, the medians $\{m(s) \mid s \in \tilde{\mathcal{S}} \wedge \max(s) = T(v)\}$ can be computed in $O(n)$ time.*

Proof By Lemma 5, $m(s_j)$ lies in subtree $T(v_j)$ with the largest $\overline{W}_t[v_j]$ if $s_j \in \tilde{\mathcal{S}}$, and by Lemma 6, $m(s)$ lies on $\pi(m(s_j), v)$ if $s \in \tilde{\mathcal{S}}$. To identify the vertex $m(s)$, starting from $m(s_j)$, we test each vertex on $\pi(m(s_j), v)$ until the condition of Lemma 1 is satisfied. We perform a post-order traversal of T with some additional steps. When v is visited in the post-order traversal, we compute the maximum weight of its subtrees. A vertex u may be visited for the second time if it is on $\pi(m(s_j), v)$, where v_j is a child of v, when v is visited. We charge this time to v, unless the median under s with $f(s) = v$ moves up from $m(s_j)$, in which case we charge it to $m(s_j)$. Thus the total time required is $O(n)$. □

3.2 Computing Median $m(s)$ for Some $s \in \tilde{\mathcal{S}}$ with $\min(s) = T(v)$

Let $s^c(v)$ denote the scenario such that $\min(s^c(v)) = T(v)$. As commented earlier, computing $\{m(s) \mid s \in \tilde{\mathcal{S}}\}$ takes $\Omega(n \log n)$ time [7]. Fortunately, as we will show, we will need $m(s^c(v))$ for only some vertices v. For simplicity, we assume that T is a binary tree. If not, we can easily convert it into one by introducing $O(n)$ dummy vertices of weight 0 [15]. Let v_a (resp. v_b) denote the left (resp. right) child of root r_0 of T. It is easy to prove

Lemma 7 *Assume that $s^c(v_a) \in \tilde{\mathcal{S}}$. For any vertex $v \in T(v_a)$, if $m(s^c(v)) \notin T(v_a)$, then $m(s^c(v)) \in \pi(m(s^c(v_a)), r_0)$.*

Let s_T denote the scenario such that $\max(s_T) = T$. Without loss of generality we assume that $m(s_T) \in T(v_a)$. Under s_T, we define the *spine* of T as a path $\langle r_0, v_1 = v_a, v_2, \ldots, v_z \rangle$, where v_z is a leaf vertex, through $T(v_a) \cup \{r_0\}$ as follows. For $i = 1, \ldots, z-1$, let v'_i be the child vertex of v_i that is not on the spine. See Fig. 2. Under s_T, the weight of $T(v_{i+1})$ is not smaller than that of $T(v'_i)$.

Lemma 8 *Let $\langle r_0, v_1, v_2, \ldots, v_z \rangle$ be the spine of T. The medians $\{m_i \mid i = 1, \ldots, z+1\}$ can be computed in $O(n)$ time, where $m_i = m(s^c(v_i))$ for $i = 1, \ldots, z$ and $m_{z+1} = m(s_T)$.*

Proof Under one specific scenario such as $s^c(v_a)$, we can find $m(s^c(v_a)) = m_1$ in $O(n)$ time. Assume that $m_1 \in T(v_b) \cup \{r_0\}$ and let k be the largest index such that m_k lies on $\pi(m_1, r_0)$. (Lemma 7.) Based on Lemma 6, we can find all the medians

$m_2, \ldots, m_k$ that lie on $\pi(m_1, r_0)$ in $O(n)$ time, performing the w-centroid test (4) on the vertices on $\pi(m_1, r_0)$ under $s^c(v_i)$ for $i = 1, \ldots, k$. If $k < z$, then $m_{k+1}, \ldots, m_z$ lie in $T(v_a)$. Let us now find them as well as m_{z+1}. From the definition of the spine $\pi(r_0, v_z)$, it is easy to see that m_{z+1} lies on $\pi(r_0, v_z)$. If we collapse $T(v_b)$ into a "super-vertex" with the weight equal to the total weight of the vertices in it, and consider v_z as the root of the new tree, we have a situation as in Theorem 1, except that only the medians under the scenarios s such that $f(s) \in \pi(m_{k+1}, v_z)$ and $r_0 \in \max(s)$ need be computed. Therefore, it can easily be done in $O(n)$ time. □

Suppose we have computed $\{m_i \mid i = 1, \ldots, z+1\}$ by Lemma 8, and let $m_{z+1} = v_g$. See Fig. 2. For subtrees $T(v'_g), T(v'_{g+1}), \ldots, T(v'_{z-1})$, compare the differences $\overline{W}_t[v'_i] - \underline{W}_t[v'_i]$ $(g \le i \le z-1)$, and let $T(v'_h)$ have the largest difference $\overline{W}_t[v'_h] - \underline{W}_t[v'_h]$. Restarting at r_0, and following the subtrees with the largest weights under $s^c(v'_h)$, we can find $m(s^c(v'_h))$ in $O(n)$ time. In Fig. 2, it is shown to lie in $T(v'_j)$.

Lemma 9

(a) *For any $v \in T(v'_i)$, where $1 \le i < g$, $m(s^c(v))$ lies on $\pi(m_{z+1}, v_z)$ (see the dashed path in Fig. 2).*
(b) *For any $v \in T(v'_i)$, where $g \le i \le z$, $m(s^c(v))$ lies on $\pi(m(s^c(v'_h)), m_{z+1})$ (see the chained path in Fig. 2).*

Proof Let v be any vertex in $T(v'_i)$. See Fig. 2. The only difference between s_T and $s^c(v)$ is the weight of $T(v)$. Clearly, $T(v)$ is not heavier under $s^c(v)$ than under s_T.

(a) $[1 \le i < g]$ As v moves towards the spine, $T(v)$ grows, T under $s^c(v)$ becomes lighter, and $m(s^c(v))$ may shift from m_{z+1}, but will stay on the spine.

(b) $[g \le i \le z]$ As v moves from v'_i towards a leaf of $T(v'_i)$, the weight of $T^c(v)$ increases. Thus the median will move towards m_{z+1}. To show that the median moves along $\pi(m(s^c(v'_h)), m_{z+1})$, let us compare the processes of finding $m(s^c(v'_h))$ and $m(s^c(v))$, starting at r_0. (We don't actually need to do this to find $m(s^c(v))$. There is a more efficient way.) Up to vertex v_j, we follow the same path under $s^c(v'_h)$ and $s^c(v)$. At v_j, $T(v_{j+1})$ may be heavier than $T(v'_j)$ under $s^c(v)$, in which case $m(s^c(v)) \in \pi(v_j, m_{z+1})$. So assume that $T(v'_j)$ is heavier than $T(v_{j+1})$ under $s^c(v)$ as well. In this case, we enter $T(v'_j)$ under both scenarios. Whenever we visit the next vertex u in $T(v'_j)$, the only difference between $s^c(v)$ and $s^c(v'_h)$ is the weight of $T^c(u)$, and the weights of the subtrees of u are the same. Thus the same path is taken under the two scenarios. The only difference is that $m(s^c(v))$ may be found before $m(s^c(v'_h))$. □

Let us now actually find $m(s^c(v))$ in Case (a) of Lemma 9. Recall that $m_{z+1} = m(s_T)$. Under scenario $s^c(v)$, all the vertices in $T(v)$ take their minimum weights, making $T(v)$ lighter than under s_T. This may put $m(s^c(v))$ away from $m_{z+1} = v_g$ towards v_z. Starting from v_g along $\pi(v_g, v_z)$, we can test each vertex against the condition in (4) to find $m(s^c(v))$.

What we actually need later is to find $m(s)$ for all $s \in \tilde{\mathcal{S}}$ with $\min(s) = T(u)$ for some vertex u. In other words, we want to compute $m(s^c(v))$ for each v such that

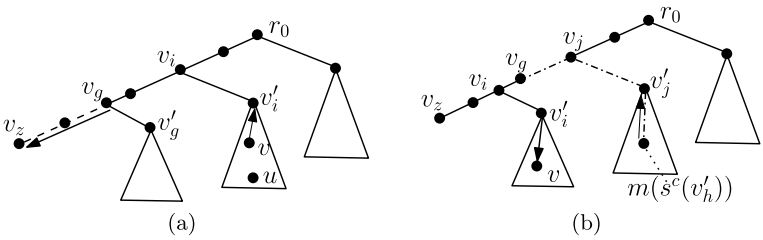

Fig. 3 Computing $m(s^c(v))$ for $v \in T(v'_i)$: (**a**) $1 \le i < g$; (**b**) $g \le i \le z$

$T(u)$ is a subtree of $T(v)$. If u is on the spine, Lemma 8 provides the solution. So, let $u \in T(v'_i)$ as in Fig. 3(a). Imagine that v moves from u towards v'_i along $\pi(u, v'_i)$. As we observed in the previous paragraph, $m(s^c(v))$ may move from $m_{z+1} = v_g$ towards v_z.

To find $m(s^c(v))$ for $v \in \pi(u, v'_i)$ systematically in an efficient way, we construct an array $D[0:t]$, where $t = z - g$, as follows. For $j = 0, 1, \ldots, z - g$, we set $D[j]$ to the minimum amount of weight reduction from s_T in $T^c(v_g)$ that causes the median to move to v_{g+j}. Note that as v moves towards v'_i along $\pi(u, v'_i)$, the weight of $T^c(v_g)$ gets reduced. Clearly, the elements of $D[\cdot]$ are in the increasing order. It is easy to prove

Lemma 10 *Let $D[0:t]$ be as defined above. For a vertex $v \in \pi(u, v'_i)$, let d be the index satisfying*

$$D[d] \le \overline{W}_t[v] - \underline{W}_t[v] < D[d+1].$$

Then $m(s^c(v))$ is the dth vertex from v_g on $\pi(v_g, v_z)$.

In Case (b) of Lemma 9, we can find $m(s^c(v))$ by a similar method, referring to Fig. 3(b). In this case array $D[\cdot]$ is constructed for the vertices on $\pi(m(s^c(v'_h)), m_{z+1})$. Before ending this subsection, we state a simple fact as a lemma.

Lemma 11 *Let tree T be rooted at vertex r_0. For any vertex $u \in T$, each scenario in $\{s \in \tilde{\mathcal{S}} \mid u \in \min(s)\}$ is either of the following two kinds.*

(a) $\max(s) = T(v)$ *for some vertex* $v \notin \pi(u, r_0)$.
(b) $s = s^c(v)$, *where* $v \in \pi(u, r_0)$.

Note that the medians under all the scenarios of the kind (a) are covered by Theorem 1, while the medians under some of the scenarios of the kind (b) are covered by Lemma 8. We will compute the medians of some of the remaining scenarios later, as we need them.

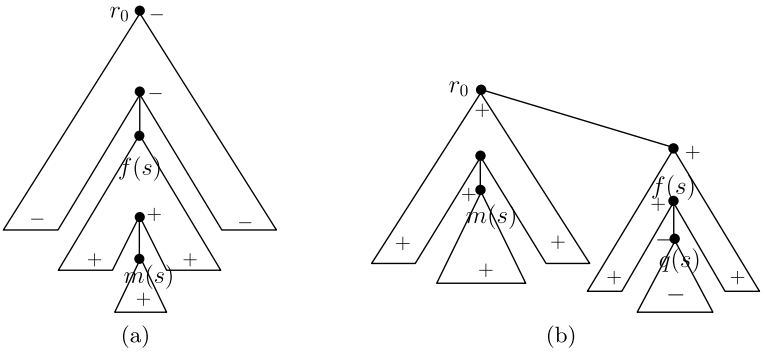

Fig. 4 Illustration for the proof of Lemma 12: (**a**) Case (a); (**b**) Case (b)

3.3 Computing Costs of Medians

We first define the *subtree costs* (with subscript t) and *complement costs* (with subscript c) relative to the root r as follows:

$$\overline{C}_t[v] = \sum_{u \in T(v)} d(u,v)\overline{w}_u, \qquad \underline{C}_t[v] = \sum_{u \in T(v)} d(u,v)\underline{w}_u,$$

$$\overline{C}_c[v] = \sum_{u \in T^c(v)} d(u,v)\overline{w}_u, \qquad \underline{C}_c[v] = \sum_{u \in T^c(v)} d(u,v)\underline{w}_u.$$

Arrays $\overline{C}_t[\cdot]$, $\underline{C}_t[\cdot]$, $\overline{C}_c[\cdot]$, and $\underline{C}_c[\cdot]$ can be computed in $O(n)$ time. (See Sect. 2.3 of [6].)

Lemma 12 *Given the median $m(s)$ under a scenario $s \in \tilde{\mathcal{S}}$, we can compute its cost $F^s(m(s))$ in constant time.*

Proof Case (a) ($m(s)$ and $f(s)$ belong to the same subtree under the root r_0, as in Fig. 4(a)): We can compute $m(s)$ for all such $s \in \tilde{\mathcal{S}}$ in $O(n)$ time by Theorem 1. We now compute their costs $F^s(m(s))$ in two possible cases. In Fig. 4, vertex v with weight $\overline{w}_v$ (resp. $\underline{w}_v$) is indicated by a $+$ (resp. $-$). Let us consider cost contributions to $F^s(m(s))$ from different parts of tree T.

1. From $T(m(s))$: $\overline{C}_t[m(s)]$.
2. From $T(f(s)) \setminus T(m(s))$: $\overline{C}_c[m(s)] - \{\overline{C}_c[f(s)] + d(f(s), m(s))\overline{W}_c[f(s)]\}$.
3. From $T^c(f(s))$: $\underline{C}_c[f(s)] + d(f(s), m(s))\underline{W}_c[f(s)]$.

It is clear that, using arrays $\overline{C}_*[\cdot]$, $\underline{C}_*[\cdot]$, $\overline{W}_c[\cdot]$, and $\underline{W}_c[\cdot]$, where $* \in \{t, c\}$, we can compute the above three in constant time.

Case (b) ($m(s)$ and $f(s)$ do not belong to the same subtree under the root r_0, as in Fig. 4(b)): We can similarly compute $F^s(m(s))$ in constant time. □

4 Optimal Facility Location

Now that we have efficient methods to compute medians and their costs, we shall address the main problem of finding the optimal facility location. We first describe our general approach, then present our algorithm, and finally analyze its complexity.

4.1 Prune and Search

Scenario s is said to be *dominated in* a subtree, if for every point x in the subtree, there is a scenario that dominates s at x. Given a tree T, as in [5, 16], we successively identify smaller and smaller part of T, named T^*, that contains the optimal location x^*. In addition, we maintain a shrinking set S, initialized to $\tilde{\mathcal{S}}$, of scenarios that may not be dominated in T^*, while those in $\tilde{\mathcal{S}} \setminus S$ are known to be dominated. The main issue is that the fronts of the scenarios in S and their medians may belong to $T \setminus T^*$. To keep track of all the scenarios in S, we maintain a tree $T' = (N, E')$, called the *auxiliary tree*, which contains T^* in it. See Fig. 5.

The node set[1] N of T' consists mainly of the front vertices of the scenarios in S. We define the *scenario-weight* of a node $u \in T'$ to be the number of scenarios in S that have u as their fronts. A node with scenario-weight 0 is called a *dummy node*, and it does not represent the front node of any scenario in S. Each node in $T' \setminus T^*$ will have a scenario-weight of no more than one, after every scenario s such that $\max(s) \supset T^*$ has been discarded. (Note that they are dominated in T^* by Lemma 3.) After that, if a scenario whose front u is in $T' \setminus T^*$ is discarded, then u becomes a dummy node. If this is a leaf node, we simply remove it from T', together with the incident edge. If a node u with degree 2 becomes dummy, we remove u and connect its neighbors u' and u'' with an edge of length $d(u, u') + d(u, u'')$. As a result, every leaf node is non-dummy, and there is no dummy node with degree 2. For example, see Fig. 5, where the hollow circles indicate the dummy nodes. Therefore, the number of dummy nodes cannot be more than the number of leaves. This implies that the number of nodes in T' is $O(|S|)$.

If we remove a point $y \in T$ and the edges incident to it from T, a number of "subtrees" result. Any such "subtree" with point y and the edge to it restored is called a y-*branch* [2] of T. We use $B_T(y, x)$ to denote the y-branch of T that contains point x ($\neq y$). We now cite a useful lemma.

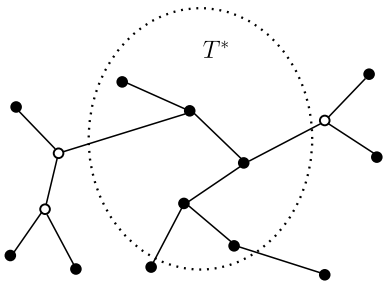

Fig. 5 Auxiliary tree T' containing T^*. *The hollow circles* represent dummy nodes

[1] We use the term 'node' to avoid confusion with a vertex of T.

Lemma 13 [2, Lemma 1] *For any point $x \in T$, the optimal location x^* is in the x-branch in which the worst case alternative $m(\hat{s}(x))$ for x lies, where $s = \hat{s}(x)$ maximizes $R^s(x)$.*

To localize x^* eventually to a point, we want to make the sizes of T^* and S smaller and smaller. To this end, we pick the *pivot vertex* $r \in T^*$, determine the r-branch of T^* that contains the optimal location x^*, and update T^* to this r-branch. We want to choose r judiciously, so that the sizes of new T^* and S will be no more than a constant fraction of their current sizes. With these considerations, we use a w-centroid of T' (based on the scenario-weights) as r, which can be found in time linear in the size of T^* [9]. By definition of the w-centroid, none of the r-branches minus r, $T^* \setminus \{r\}$ in particular, contains more than $|S|/2$ fronts, and therefore, $T' \setminus T^* \cup \{r\}$ contains at least $|S|/2$ fronts. This fact will be used in deriving (18) later.

After picking pivot vertex r, theoretically, we can determine $B_{T'}(r, x^*)$ of T', based on Lemma 13. The practical difficulty is that we need some medians that are not covered by Theorem 1 or Lemma 8. For those other medians, we need to resort to Lemma 10, but we cannot afford to spend more than $O(|S|)$ time in total, which is a challenge that we will overcome later.

4.2 Preparation

Let us define $S_{\min}(r)$ by

$$S_{\min}(r) = \left\{ s \in S \mid \min(s) \supseteq B_{T'}\left(r, x^*\right) \setminus \{r\} \right\}. \tag{10}$$

Lemma 14 [16] *Let $s \in S_{\min}(r)$. Then the cost function $F^s(x)$ over $x \in T^* \cap B_{T'}(r, x^*)$ has the following properties*:

(a) *It is a non-decreasing linear function of point x on each edge, as x moves away from r.*
(b) *It is continuous at the vertices.*

Lemma 15 *Let $s, s' \in S_{\min}(r)$. Unless $R^s(x) = R^{s'}(x)$ for all $x \in T^*$, $R^s(x) = R^{s'}(x)$ holds for at most one point $x \in \pi(r, u)$ for each leaf u of T^*.*

Proof Let $e = (v, v')$ be an edge of T^* such that v' is closer to r than v is. The rates of change of $F^s(x)$ and $F^{s'}(x)$ (hence of $R^s(x)$ and $R^{s'}(x)$) with respect to $x \in e$, as x moves towards v' are $(W^s - W^s(v)) - W^s(v) = W^s - 2W^s(v)$ and $W^{s'} - 2W^{s'}(v)$, respectively, where $W^s(v)$ is the total weight under s of the subtree $T(v)$, and W^s is the total weight under s of all the vertices of T. Their difference is thus

$$\left\{W^s - W^{s'}\right\} + 2\left\{W^{s'}(v) - W^s(v)\right\}. \tag{11}$$

The term $\{W^s - W^{s'}\}$ clearly does not depend on the edge e on which x lies. Clearly, $\{W^{s'}(v) - W^s(v)\} = 0$ holds, because $s, s' \in S_{\min}(r)$. Therefore, $R^s(x) = R^{s'}(x)$ holds for at most one point $x \in \pi(r, u)$, unless $R^s(x) = R^{s'}(x)$ for all $x \in T^*$. □

Lemma 16 *Assume that* $\overline{W}_*[\cdot]$, $\underline{W}_*[\cdot]$, $\overline{C}_*[\cdot]$, *and* $\underline{C}_*[\cdot]$ *are known, where* $* \in \{t, c\}$. *Given a set* $S \subset \tilde{S}$ *of scenarios and a point* x *such that* (i) $\forall s \in S : x \in \min(s)$, *and* (ii) $\{m(s) \mid s \in S\}$ *are known, we can determine in* $O(|S|)$ *time the scenario* $\hat{s}(x) \in S$ *that dominates all others in* S *at* x, *and the* x*-branch that contains* $m(\hat{s}(x))$, *i.e.*, $B_{T^*}(x, m(\hat{s}(x)))$.

Proof We need to compute $R^s(x) = F^s(x) - F^s(m(s))$ for each $s \in S$. We can compute $F^s(m(s))$ in $O(1)$ time by Lemma 12. To evaluate $F^s(x)$, we interchange the max-weighted vertices and min-weighted vertices ($f(s)$ moves as a result) in Fig. 4, and replace $m(s)$ by x, except that x can be any point, not necessarily a vertex. Following the method in the proof Lemma 12, it is clear that $F^s(x)$, hence $R^s(x)$, can be computed for all $s \in S$ in $O(|S|)$ time. Then $\hat{s}(x)$ is given by the scenario s that maximizes $R^s(x)$. It is easy to find $B_{T^*}(x, m(\hat{s}(x)))$. □

Regarding Condition (i) in Lemma 16, if $x \notin \min(s)$ for $s \in S$ then s is dominated by some other scenario at x, and can be ignored. Condition (ii) assumes the medians $\{m(s) \mid s \in S\}$ are available, but computing them is still a pending issue to be settled in the next section.

4.3 Algorithm

We first introduce a procedure that performs the task of Lemma 16, which runs in $O(|S|)$ time.

Procedure `OptBranch`(S, x): [$\{m(s) \mid s \in S \wedge x \in \min(s)\}$ are known]
Find the x-branch that contains the optimal location.

We start with $S = \tilde{S}$ and the initial pivot r set to the w-centroid of T', and keep track of the shrinking scenario set S and T^* that is an r-branch containing the optimal location x^*. To find this r-branch, we need to know the medians of the scenarios in $\{s \in S \mid r \in \min(s)\}$. Then we can remove from S the scenarios that are dominated in T^*. The scenarios in $\{s \in S \mid \max(s) \supseteq T^*\}$ are obviously dominated in T^* and can be discarded. Among the remaining scenarios in S, we pay special attention to those in $\{s \in S \mid \min(s) \supseteq T^*\}$.

We use the w-centroid of T as r_0. Assume that the medians of the scenarios covered by Theorem 1 have been computed, so that the medians of the scenarios in $\{s \in \tilde{S} \mid r_0 \in \min(s)\}$ are known. We also assume that the medians $\{m_i \mid i = 1, \ldots, z+1\}$ have been computed by Lemma 8. Here is an outline of our algorithm, where κ is a constant.

Algorithm `Prune`

1. Set $T^* = T$ and $S = \tilde{S}$.
2. For $j = 1, 2, \ldots,$ carry out Steps 3 to 7.
3. Pick a pivot vertex r. If r is not on the spine, compute the medians of the scenarios $\{s^c(v) \in S \mid v \in \pi(r, v_i')\}$, where $r \in T(v_i')$ in Fig. 2.

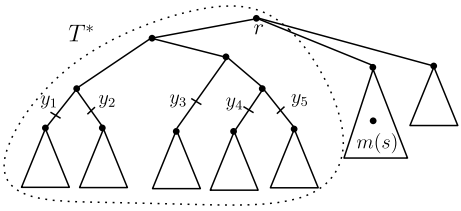

Fig. 6 Points $y_1, y_2, \ldots, y_5$ are at distance d_m from r

4. Invoke Procedure `OptBranch`(S, r) to determine the r-branch that contains the optimal location x^*. Update T^* to this r-branch.
5. Discard from S the scenarios in $\{s \in S \mid \max(s) \supseteq T^*\}$.
6. Let $S_{\min}(r)$ $(\subseteq S)$ be the set of scenarios defined by (10). Pair up scenarios in $S_{\min}(r)$, and let p be the number of pairs.[2]
7. Determine and discard at least $\lceil p/4 \rceil$ dominated scenarios from S. If $|S| \leq \kappa$, then locate x^* by an exhaustive means or by applying any of the algorithms currently available (e.g., that in [2]) and stop.

The first thing we do in a round is to pick pivot r in Step 3. As we discuss later, in general, we set the pivot r to the w-centroid of T^*. Thus in Step 3 of the first round, we have $r = r_0$. To determine the r-branch of T^* that contains the optimal location x^*, $B_{T^*}(r, x^*)$ in Step 4, we may need some preparation in Step 3, by computing the needed medians $\{m(s) \mid s \in S \wedge r \in \min(s)\}$. See Lemmas 10 and 11. But in the first round they are already known.

Since Step 5 is straightforward, let us now discuss Steps 6 and 7, which form the core of Algorithm `Prune`. Since $\{m(s) \mid s \in S \wedge r \in \min(s)\}$ are available after Step 3, then obviously $\{m(s) \mid s \in S_{\min}(r)\}$, which is needed in Step 6, are also available, except possibly for one scenario.[3] Step 7 requires, among others, computing the cross-over point of the regret functions $R^s(x)$ and $R^{s'}(x)$ of each pair (s, s') of scenarios constructed in Step 6. Note that $m(s)$ and $m(s')$ were either precomputed or computed in Step 3. The most favorable cases are where either $R^s(x) = R^{s'}(x)$ for all $x \in T^*$ or $R^s(x)$ and $R^{s'}(x)$ don't cross each other in T^*. In these cases, one of s and s' is dominated by the other in T^* and can be thrown away, achieving a 2-to-1 reduction. However, in the worst case, there may be no such pair.

Let us assume the least favorable case, where $R^s(x) = R^{s'}(x)$ has a (non-zero) finite number of solution points in T^*. See Lemma 15. All solution points to $R^s(x) = R^{s'}(x)$ lie at the same distance from r. Suppose that $d_1, d_2, \ldots, d_p$ are the distances from r of the solution points for the p pairs of scenarios. Let d_m be the $\lceil p/2 \rceil$th smallest among them, and let d^* denote the distance from r of the optimal location x^*, which is not known yet. If $d^* \geq d_m$ (resp. $d^* < d_m$), then from each scenario pair with solution point distance $d_i \leq d_m$ (resp $d_i \geq d_m$), one of them can be thrown away, because the other dominates it at any point x that is not nearer (resp. nearer) than d_m from r.

[2] If $|S_{\min}(r)|$ is odd, one scenario is left unpaired.

[3] In the first round, where $r = r_0$, for example, this scenario may be $s^c(v_a)$ or $s^c(v_b)$. In general such a scenario belongs to $S_{\min}(r)$ and its front is r. It is easy to compute its median.

We now show how to determine if $d^* \geq d_m$ for a given value d_m (> 0) without knowing the exact value of d^*. Let $y_1, y_2, \ldots, y_c$ be the points in T^* at distance d_m from r. For example, see $y_1, \ldots, y_5$ in Fig. 6. Let $\overline{T}(y_i)$ be the y_i-branch that is "below" (farther away from r) y_i. Thus $\overline{T}(y_i) = T(y_i)$ holds if y_i is a vertex. Let $S_i (\subset S)$ denote the set of scenarios such that for any $s \in S_i$ we have $\max(s) \cap T^* \subseteq \overline{T}(y_i)$. Thus their min-weighted components contain r. Define

$$\overline{R}(y_i) = \max_{s \in S_i} \{ F^s(y_i) - F^s(m(s)) \}, \tag{12}$$

and let $s_i \in S_i$ realize $\overline{R}(y_i)$. Note that we already know $\{m(s) \mid s \in S_i\}$ as a result of Step 3. So we can determine s_i in $O(|S_i|)$ time by Lemma 16. Among them let $\overline{R}(y_k)$ (realized by $s_k \in S_k$) be as large as any other. We now compute $R^*(y_k)$ and $\hat{s}(y_k)$ based on the scenarios in S ($\subset \tilde{S}$), which is the set of all scenarios in $\tilde{S}$ that have not been thrown away so far. This can be done in $O(|S|)$ time by Lemma 16, provided the relevant medians are known. The relevant medians that we don't know yet are those under the scenarios whose min-weighted (resp. max-weighted) components contain y_k (resp. r). If such a median is not known yet, we can resort to Lemma 10 with $u = y_k$.

Lemma 17 *For a given value d_m of x, define $\{y_i \mid i = 1, 2, \ldots, c\}$ and y_k as above.*

(a) *If $\hat{s}(y_k) \in S_k$ then the optimal location x^* lies in $\overline{T}(y_k)$.*
(b) *If $\hat{s}(y_k) \notin S_k$ then the optimal location x^* cannot lie in $\overline{T}(y_i)$ for any i.*

Proof (a) Follows from Lemma 13, since $\hat{s}(y_k) \in \tilde{S}$, hence $m(\hat{s}(y_k)) \in \overline{T}(y_k)$.

(b) The assertion is trivially true for $i = k$. Assume that the optimal location (over S) was in $\overline{T}(y_j)$ ($j \neq k$). Then by Lemmas 4 and 13, the scenario that realizes $R^*(y_j)$ must be in S_j, and hence $R^*(y_j) = \overline{R}(y_j)$. By the definition of k, we have

$$R^*(y_j) = \overline{R}(y_j) \leq \overline{R}(y_k). \tag{13}$$

On the other hand, we have

$$\overline{R}(y_k) < R^{s_k}(y_j), \tag{14}$$

by Lemma 14. We have strict inequality here, because y_j is farther away from median $m(s_k)$ than y_k. By definition, we also have

$$R^{s_k}(y_j) \leq R^*(y_j). \tag{15}$$

Equations (13), (14) and (15) yield $R^*(y_j) < R^*(y_j)$, a contradiction. □

Procedure `OptBranch`(S, y_k) can determine whether we have Case (a) or (b) of Lemma 17. In Case (a), we can eliminate one scenario from each pair of scenarios, s and s', whose crossover point (where $R^s(x)$ and $R^{s'}(x)$ intersect) is at most distance d_m away from r. In Case (b), we can eliminate one scenario from each pair of scenarios whose crossover point is at least distance d_m away from r. Note that to execute Procedure `OptBranch`(S, y_k), we need the medians of the scenarios

$\{s^c(v) \in S \mid v \in \pi(v_k, v_i')\}$, which can be computed by Lemma 10. However, computing them individually takes too much time. In the next subsection, we present a way to batch them to save time.

4.4 Two-Phase Tree Search

Assume that a *target sequence* of ordered data points is given as an array, $D[0:t]$, where $D[0] = 0$, and $D[i] < D[i+1]$ holds for $i = 1, \ldots, t-1$. There are also q arrays, $Q_j[1:l_j]$ of integers ($j = 1, \ldots, q$), $Q_j[i] \le Q_j[i+1]$ holds for $1 \le i < l_j$, and $\sum_{j=1}^{q} l_j \le n$. Our objective is to assign $Q_j[k]$ to $D[h]$ such that $D[h] \le Q_j[k] < D[h+1]$ for $k = 1, 2, \ldots, l_j$. A naïve approach might merge $Q_j[\cdot]$ and $D[\cdot]$ into one ordered sequence for each j, which takes

$$O\left(\sum_{j=1}^{q}\{l_j + t\}\right) = O\left(qt + \sum_{j=1}^{q} l_j\right) = O(qt + n) \tag{16}$$

time. Lemma 18 will show that with one-time preprocessing, which takes $O(t)$ time, it can be done in

$$O\left(\sum_{j=1}^{q}\{l_j \log(t/l_j) + l_j\}\right) + O(t) \tag{17}$$

time. At first glance, (17) may not appear any better than (16), but if $l_i \le cn\alpha^i$, where c and α are constants satisfying $c > 0$ and $0 < \alpha < 1$, then (17) becomes $O(n)$, as proved in Theorem 2.

In preprocessing, we construct a balanced binary tree τ_D on t leaves representing $D[0], D[1], \ldots, D[t]$, from left to right. Given a *query array* $Q_j[\cdot]$ of ascending integers, we want to assign $Q_j[k]$ to $D[h]$ such that $D[h] \le Q_j[k] < D[h+1]$. During the construction of τ_D, we label each internal node[4] u by $d[u]$, which is the value of the leftmost leaf of the subtree rooted at u. Tree τ_D can be constructed in $O(t)$ time. Let $\tau_D(u)$ denote the subtree of τ_D rooted at u.

Algorithm 2PTS($D[0:t]$, $Q[1:l]$)

- *Phase 1*: If $\lceil \log l \rceil > \lfloor \log t \rfloor$,[5] then for $k = 1, 2, \ldots, l$ find h_k such that $D[h_k] \le Q[k] < D[h_k + 1]$ by merging two ordered arrays $D[\cdot]$ and $Q[\cdot]$, and stop. Otherwise, let $l' = 2^{\lceil \log l \rceil}$. Identify all the nodes, $u_1, u_2, \ldots, u_{l'}$, of τ_D that are at level (=depth) $\lceil \log l \rceil$. For $k = 1, 2, \ldots, l$ find i such that $d[u_i] \le Q[k] < d[u_{i+1}]$, by merging the sequence $\langle d[u_1], d[u_2], \ldots, d[u_{l'}]\rangle$ and the elements in the ordered array $Q[\cdot]$.
- *Phase 2*: For each $k = 1, 2, \ldots, l$, move down in $\tau_D(u_i)$ to reach the leaf node $D[h_k]$ such that $D[h_k] \le Q[k] < D[h_k + 1]$.

[4] We use the term 'node' here to distinguish it from a vertex of tree T.

[5] Note that all leaves of τ_D are at level $\lfloor \log t \rfloor$ or higher (closer to the root).

Lemma 18 *Algorithm* 2PTS$(D[0:t], Q[1:l])$ *runs in* $O(l\log(t/l)+l)$ *time.*

Proof It is clear that 2PTS$(D[0:t], Q[1:l])$ runs in $O(l)$ time if $\lceil\log l\rceil > \lfloor\log t\rfloor$. So, assume $\lceil\log l\rceil \le \lfloor\log t\rfloor$. In Phase 1, we test $Q[k]$ against the value stored at successive u_i without backtracking. This takes $O(l+l') = O(l)$ time, and each element $Q[k]$ has been assigned to a node at level $\lceil\log l\rceil$. In Phase 2, the length of a path from level $\lceil\log l\rceil$ down to a leaf is at most $\lceil\log t\rceil - \lceil\log l\rceil = O(\log(t/l))$. For all the l elements in $Q[\cdot]$, the total time is thus $O(l\log(t/l))$. □

4.5 Time Complexity Analysis

To make our analysis easier, we start with $\mathcal{S}^*$ instead of $\tilde{\mathcal{S}}$. Let us first consider Round 1, in which we have just found a w-centroid r_0 in the auxiliary tree $T' = T$ and identified T^*. For each leaf u of T there is just one scenario in $\mathcal{S}^*$, whose max-weighted component consists of just u. Let γ be the total number of scenarios in $\mathcal{S}^*$ whose min-weighted components contain T^*, and let δ be the total number of scenarios in $\mathcal{S}^*$ whose max-weighted components contain T^* (hence r). Note that for each node v, there are $d(v)$ scenarios in $\mathcal{S}^*$ whose fronts are on v, where $d(v)$ denotes the degree of v. For node v in $T \setminus T^*$, among those $d(v)$ scenarios, all but one contain T^* (hence r) in their max-weighted components. At any point $x \in T^*$, these $d(v)-1$ scenarios are dominated by other scenarios by Lemma 3. Since T^* keeps shrinking over the successive rounds, this dominance relation does not change in the future, which implies that we can throw them away for good.

The total number of scenarios in $\mathcal{S}^*$ whose fronts are on the nodes in $T \setminus T^* \cup \{r_0\}$ is given by

$$\gamma + \delta + 1 \ge |S|/2, \tag{18}$$

where the inequality holds because r_0 is a w-centroid. On the left hand side of (18), "+1" accounts for the scenario s with $\min(s) = T^* \setminus \{r\}$. Step 4 (resp. Step 7) of Algorithm Prune throws away δ (resp. $\gamma/8$) scenarios from S. We have

$$\begin{aligned}\delta + \gamma/8 &= (7/8)\delta + (\gamma+\delta+1)/8 - 1/8\\ &= (7\delta-1)/8 + (\gamma+\delta+1)/8\\ &\ge |S|/16,\end{aligned} \tag{19}$$

since $\delta > 1$. Therefore, Algorithm Prune removes at least $|S|/16$ scenarios, reducing the size of S by at least $1/16$ in each round. Thus, after round j, the size of S reduces to less than $2n(15/16)^j$. Note that $|\mathcal{S}^*| = 2(n-1)$, because each edge of T gives rise to two scenarios in $\mathcal{S}^*$.

Assume that the medians of the scenarios covered by Theorem 1 and the medians $\{m_i \mid i = 1,\ldots,z+1\}$ have been computed by Lemma 8. To execute OptBranch(S,r) (resp. OptBranch(S,y_k)) invoked in Step 4 (resp. Step 7) of Algorithm Prune, we need some other medians. We first discuss the medians needed to execute OptBranch(S,r). If r lies on the spine, then we already know the medians of all scenarios in $\{s \in S \mid r \in \min(s)\}$. So, let $u = r \in T(v_i')$ in Fig. 3. We

will only discuss how to find the medians covered by Lemma 9(a), since those covered by Lemma 9(b) can be computed similarly. For this purpose, we introduced Algorithm 2PTS($D[0:t]$, $Q[1:l]$) in Sect. 4.4. To use it, we need the target sequence $D[0:t]$ associated with the vertices on $\pi(m_{z+1}, v_z)$, which we constructed just before Lemma 10. We then construct a search tree τ_D, which is introduced in Sect. 4.4.

As for the *query array* $Q[1:l]$, let v be the jth vertex (counting from r) along $\pi(r, v_i')$ in the set $\{v \in \pi(r, v_i') \mid s^c(v) \in S\}$. We set $Q[j] = \overline{W}_t[v] - \underline{W}_t[v]$. (See Lemma 10.) Clearly, the elements of $Q[\cdot]$ are in the non-decreasing order. We can now execute Algorithm 2PTS($D[0:t]$, $Q[1:l]$), to locate $\{m(s^c(v)) \mid v \in \pi(r, v_i') \wedge s^c(v) \in S\}$.

The medians needed for OptBranch(S, y_k) can be computed similarly, considering path $\pi(y_k, v_i')$ instead of $\pi(r, v_i')$.[6] The *S-size* of a path π is defined to be the number of vertices in $\{v \in \pi \mid s^c(v) \in S\}$. We now prove an important theorem.

Theorem 2 *Let $\pi_1, \pi_2, \ldots, \pi_q$ be disjoint paths in $T(v_a)$ of S-sizes, $l_1, l_2, \ldots, l_q$, respectively, such that for each j, π_j is a subpath of a path that starts at r_0. Under the conditions that $l_j \le cn\alpha^j$ for each j, where c and α are constants satisfying $c > 0$ and $0 < \alpha < 1$, and n is the size of tree T, we can find $m(s^c(v))$ for all vertices $v \in \pi_j$ for all j in $O(n)$ time.*

Proof In Phase 1 of Algorithm 2PTS($D[0:t]$, $Q[1:l_j]$), we can use level $\lceil cn\alpha^j \rceil$ of τ_D in Phase 1. Then by Lemma 18, the medians under the scenarios in $\{s^c(v) \in S \mid v \in \pi_j\}$ can be computed in $O(l_j \log(t/\lceil cn\alpha^j \rceil))$ time after some preprocessing (construction of $D[0:t]$ and τ_D) that costs $O(t)$ time. Thus the total time required by Algorithm 2PTS() to compute $m(s^c(v))$ for all j and for all vertices $v \in \pi_j$ is given by

$$
\begin{aligned}
&O\left(\sum_{j=1}^{q} l_j \log\bigl(t/\lceil cn\alpha^j \rceil\bigr) + l_j\right) \\
&\le O\left(\sum_{j=1}^{q} \bigl(c\alpha^j n\bigr) \log\bigl(\alpha^{-j}/c\bigr)\right) + O(n) \\
&= O\left(nc\log(1/\alpha) \sum_{j=1}^{q} j\alpha^j - nc\log c \sum_{j=1}^{q} \alpha^j\right) + O(n) \\
&= O(n), \qquad (20)
\end{aligned}
$$

since $t < n$ and $\sum_{j=1}^{q} j\alpha^j < \alpha/(1-\alpha)^2$ holds independently of q, if $0 < \alpha < 1$. □

[6] If these two paths intersect, then some medians needed for OptBranch(S, y_k) may have already been computed, and they need not be recomputed.

In applying Theorem 2, we set $\pi_j = \pi(r, v_i')$ for the jth round of Algorithm `Prune`. Here we define l_j as the S-size of π_j. The paths, such as $\pi(r, v_i')$, satisfy the condition of Theorem 2 with $l_j \leq cn(15/16)^j$, since $l_j < |S| \leq cn(15/16)^j$.

Let $|S| = k$ and let $t(k)$ be the time needed for the repeated executions of Algorithm `Prune` to find the optimal location, starting with the scenarios in S. Since the execution of Algorithm `Prune` once takes linear time, the recurrence relation on $t(k)$ is

$$t(k) \leq t(15k/16) + O(k). \tag{21}$$

This recurrence equation has the solution of the form $t(k) = O(k)$. Once k becomes smaller than some constant, we can solve the problem exhaustively. We thus have the main result of this paper:

Theorem 3 *The minmax regret* 1*-median of a tree can be found in* $O(n)$ *time, where* n *is the number of vertices in the given tree.*

5 Conclusion and Future Work

We have presented an $O(n)$ time algorithm for computing the minmax regret 1-median for trees with nonnegative vertex weights. This improves upon the previously known best complexity of $O(n \log n)$, and is the best possible. Our next goal is to design an algorithm for a cactus network. A more challenging problem is how to compute the minmax regret p-median for various families of networks. We believe that some techniques that we have developed in this paper will be useful in solving other facility location and minmax problems.

Acknowledgements We greatly appreciate the comments provided by the anonymous referees, who pointed out some deficiencies in the original manuscript, and who patiently perused the two revisions of this relatively long paper. This work was supported in part by Discovery Grants of the Natural Sciences and Engineering Research Council of Canada, held by B. Bhattacharya and T. Kameda, and a MITACS grant held by B. Bhattacharya.

References

1. Averbakh, I., Berman, O.: Minmax regret median location on a network under uncertainty. INFORMS J. Comput. **12**(2), 104–110 (2000)
2. Averbakh, I., Berman, O.: An improved algorithm for the minmax regret median problem on a tree. Networks **41**, 97–103 (2003)
3. Bhattacharya, B., Kameda, T.: Linear time algorithm for finding minmax regret 1-median on a tree with positive vertex weights. In: Proc. 18th Annual International Computing and Combinatorics Conference (COCOON). LNCS, vol. 7434, pp. 1–12. Springer, Berlin (2012)
4. Bhattacharya, B., Kameda, T., Song, Z.: Computing minmax regret 1-median on a tree network with positive/negative vertex weights. In: Proc. 23rd Int'l Symp. on Algorithms and Computation (ISAAC). LNCS, vol. 7676, pp. 588–597. Springer, Berlin (2012)
5. Brodal, G.S., Georgiadis, L., Katriel, I.: An $O(n \log n)$ version of the Averbakh–Berman algorithm for the robust median of a tree. Oper. Res. Lett. **36**, 14–18 (2008)
6. Burkard, R.E., Dollani, H.: Robust location problems with pos/neg weights on a tree. Networks **38**(2), 102–113 (2001)

7. Chan, C.Y., Ku, S.C., Lu, C.J., Wang, B.F.: Efficient algorithms for two generalized 2-median problems and the group median problem on trees. Theor. Comput. Sci. **410**, 867–876 (2009)
8. Chen, B., Lin, C.S.: Minmax-regret robust 1-median location on a tree. Networks **31**, 93–103 (1998)
9. Goldman, A.: Optimal center location in simple networks. Transp. Sci. **5**, 212–221 (1971)
10. Hale, T.S., Moberg, C.R.: Location science research: a review. Ann. Oper. Res. **123**, 21–35 (2003)
11. Kariv, O., Hakimi, S.: An algorithmic approach to network location problems, part 2: The p-median. SIAM J. Appl. Math. **37**, 539–560 (1979)
12. Kouvelis, P., Vairaktarakis, G., Yu, G.: Robust 1-median location on a tree in the presence of demand and transportation cost uncertainty. Tech. Rep. Working paper 93/94-3-4, Department of Management Science, The University of Texas, Austin (1993)
13. Kouvelis, P., Yu, G.: Robust Discrete Optimization and Its Applications. Kluwer Academic, London (1997)
14. Megiddo, N.: Linear-time algorithms for linear-programming in R^3 and related problems. SIAM J. Comput. **12**, 759–776 (1983)
15. Tamir, A.: An $O(pn^2)$ algorithm for the p-median and the related problems in tree graphs. Oper. Res. Lett. **19**, 59–64 (1996)
16. Yu, H.I., Lin, T.C., Wang, B.F.: Improved algorithms for the minmax-regret 1-center and 1-median problem. ACM Trans. Algorithms **4**(3), 1 (2008)