Minmax Regret 1-Center on a Path/Cycle/Tree

Binay Bhattacharya, Tsunehiko Kameda, and Zhao Song
School of Computing Science, Simon Fraser University
University Drive, Burnaby, Canada V5A 1S6
Email: {binay, tiko, zhaos}@sfu.ca

***Abstract*—In a facility location problem in computational geometry, if the vertex weights are uncertain one may look for a "robust" solution that minimizes "regret." The best previously known algorithm for finding the minmax regret 1-center in a tree with positive vertex weights is due to Yu *et al.*, and runs in sub-quadratic asymptotic time in the number of vertices. Assuming that the minimum weight of at least one vertex is non-negative, we present a new, conceptually simpler algorithm for a tree with the same time complexity, as well as an algorithm that runs in linear (respectively sub-quadratic) time for a path (respectively cycle).**

***Index Terms*—facility location; 1-center; minmax regret optimization**

I. Introduction

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 [1]. The cost of a vertex is formulated as the distance from the nearest facility weighted by the vertex weight. In the *minmax regret* version of this problem, there is uncertainty in the weights of the vertices and/or edge lengths, and only their ranges are known. Chen and Lin (Theorem 1 in [2]) proved that in solving this problem, the edge lengths can be set to their maximum values, when the vertex weights are non-negative. Our model assumes that the edge lengths are fixed and uncertainty is only in the weights of the vertices.

A particular *realization* (assignment of a weight to each vertex) is called a *scenario*. Intuitively, the minmax regret 1-center 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 1-center, 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).

The rest of the paper is organized as follows. Section II reviews work relevant to our problem. In Section III, we define the terms that are used throughout the paper, and cite or prove some center-related properties. Section IV first discusses how to compute the centers and their costs under a certain set of scenarios. We then present a linear time algorithm for computing the minmax regret 1-center of a path. In Section V, we present an algorithm for a cycle. Finally, Section VI presents a simplified algorithm for a tree, and Section VII concludes the paper.

II. Related Work and Our Objectives

The problem of finding the minmax regret 1-center on a network, and a tree in particular, has been attracting great research interest in recent years. Several researchers have worked on this problem. The classical p-center problem is discussed by Kariv and Hakimi in [3]. Megiddo [4] computes the classical 1-center of a tree with non-negative vertex weights in $O(n)$ time, where n is the number of vertices. Averbakh and Berman [5] proved some basic results in the minmax regret 1-center problem. More recently, Yu et al. [6] solved the problem in $O(mn \log n)$ (resp. $O(n \log^2 n)$) time for a general network (resp. tree network) with positive vertex weights, where m is the number of edges.

Since the known algorithm for a general network is relatively inefficient, we look for more efficient algorithms for special families of networks. Yu et al. [6] proposed an $O(n \log^2 n)$ time algorithm for a tree with positive vertex weights. Their algorithm is rather involved, so we want to come up with a conceptually simpler algorithm, under the more relaxed assumption that at least one vertex has the maximum weight that is positive. A cycle is the simplest building block of any network that is more general than a tree, but to the best of our knowledge, nothing is known about the complexity of finding the minmax regret 1-center on a cycle. We want to design an efficient algorithm that is specifically tailored to a cycle network.

III. Preliminaries

As stated above, we assume that there is at least one vertex whose minimum weight is non-negative. Then it is clear that all vertices whose weights are negative can be ignored, because they cannot influence the 1-center. Therefore, we assume without loss of generality that the minimum weights of all vertices are non-negative.

Let $G = (V, E)$ be a path, cycle or tree network with n vertices. We also use G to denote the set of all points (vertices and points on edges) on G. Each vertex $v \in V$ is associated with an interval of 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). We assume that the distances between a point on an edge and its end vertices are prorated fractions of the edge length. For any pair of points $p, q \in G$, the shortest distance between them is denoted by $d(p, q)$. Let $\mathcal{S}$ be the Cartesian product of all $W(v),\ v \in V$:

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

The *cost* of a point $x \in G$ *with respect to* $v \in V$ under scenario s is $d(v,x)w_v^s$, where w_v^s denotes the weight of v under s, and the *cost* of x under s is defined by

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

The point x that minimizes $F^s(x)$ is called a *(classical) 1-center* under s, and is denoted by $c(s)$. Throughout this paper the term *center* refers to this weighted 1-center. The difference

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

is called the *regret* of x under s. We finally define the *maximum regret* of x as

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

The scenario that maximizes $R^s(x)$ for a given $x \in G$ is called the *worst case scenario* for x. Note that $R^*(x)$ is the maximum payoff with respect to x that we mentioned in the Introduction. We seek location $x^* \in G$, called the *minimum regret 1-center*, that minimizes $R^*(x)$.

Let $V = \{v_1, v_2, \ldots, v_n\}$. For simplicity, we use $w_i = w_{v_i}$ in what follows. The *base scenario*, s_0, is defined by $w_i^{s_0} = \underline{w}_i$ for all i. A vertex v that maximizes (1) is said to be a *critical* vertex for x [5]. For $i = 1, 2, \ldots, n$, let us define the *single-max scenario* s_i by $w_i = \overline{w}_i$ and $w_j = \underline{w}_j$ for all $j \neq i$, and let $\mathcal{S}^*$ denote the set of all single-max scenarios. Averbakh and Berman proved the following fundamental theorem for the trees, but the same proof is valid for the general network.

Theorem 1: [5] For any point x in a network, there is a worst case scenario $s_i \in \mathcal{S}^*$. Vertex v_i is a critical vertex for x. ■

Theorem 1 implies

$$R^*(x) = \max_{s \in \mathcal{S}^*} R^s(x). \tag{4}$$

Therefore, we need to consider only the single-max scenarios in looking for the minmax regret location x^*.

The cost of $x \in G$ with respect to v_i is given by

$$f_i(x) \triangleq d(x, v_i)w_i. \tag{5}$$

It will turn out that we need to consider either

$$\underline{f}_i(x) = d(x, v_i)\underline{w}_i \text{ or } \overline{f}_i(x) = d(x, v_i)\overline{w}_i, \tag{6}$$

which is called the *min-weight cost* and *max-weight cost* with respect to v_i, respectively. To evaluate (2) for $s = s_j$, we need to find $c(s_j)$ first. Most of our effort will be on how to compute $c(s_j)$ efficiently.

IV. Path Network

We start with a path, which is the simplest network.

A. Computing the Upper Envelope of a Set of Line Segments

Assume that the vertices of a path $\mathcal{P} = (V, E)$ are laid out horizontally, in the order $v_1, v_2, \ldots, v_n$ from left to right. Let $L_i(x)$ represent an increasing function, such as $\underline{f}_i(x)$, defined for x to the right of v_i. Its value gets larger as x moves right from v_i. To find $c(s_j)$ efficiently, we construct the upper envelope of $\{L_i(x) \mid i = 1, 2, \ldots, n\}$, assuming $w_1 \geq 0$. If $w_i \geq w_j$ for $i < j$, then we have $L_i(x) > L_j(x)$ for all x (to the right of v_j, where $L_j(x)$ is defined), and $L_j(x)$ won't be a part of the upper envelope. In this case, we say that $L_i(x)$ *dominates* $L_j(x)$, and can ignore $L_j(x)$. Therefore, the following algorithm assumes that $w_i < w_j$ holds for any i and j such that $i < j$.

Algorithm `Find-Envelope/Path`

```
1) Push $[1:(v_1, 0)]$ in stack $\Sigma$. (Meaning: $L_1(x)$, starting
   at $(v_1, 0)$, is the initial part of the upper envelope.)
2) For $k = 2, 3, \ldots, n$, carry out steps 3 to 5.
3) Let $[i:(x_i, y_i)]$ be the item at the top of $\Sigma$. If line $L_k(x)$
   passes above $(x_i, y_i)$ pop up the top item from $\Sigma$ and
   repeat this step.
4) Compute the intersection $(x_k, y_k)$ of $L_k(x)$ and $L_i(x)$.
   If $x_k$ is to the left of $v_n$, then push $[k:(x_k, y_k)]$ into $\Sigma$.
   ■
```

Lemma 1: When Algorithm `Find-Envelope/Path` completes, an entry $[i:(x_i, y_i)]$ in stack Σ means $L_i(x)$ forms a segment of the upper envelope of $\{L_j(x) \mid j = 1, 2, \ldots, n\}$, starting at point (x_i, y_i). `Find-Envelope/Path` runs in $O(n)$ time.

Proof: The first part is obviously true of the entry $[1:(v_1, 0)]$, which is the bottom entry of Σ that is never removed. Suppose $[j:(x_j, y_j)]$ (resp. $[i:(x_i, y_i)]$) was the entry at the top (resp. 2nd from the top) of Σ, when line $L_k(x)$ is processed. See Figure 1. If $L_k(x)$ is like the dotted line, then step 3

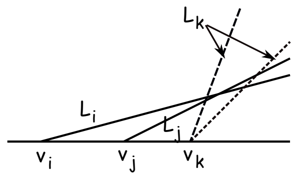

Fig. 1. Computing the upper envelope of lines $L_i(x), L_j(x)$, and $L_k(x)$.

computes the intersection of $L_k(x)$ and $L_j(x)$ and pushes $[k:(x_k, y_k)]$ into Σ, indicating that $L_k(x)$ is the next segment starting at (x_k, y_k). If $L_k(x)$ is like the dashed line, then step 4 computes the intersection $L_k(x)$ and an earlier line $L_i(x)$, and pushes $[k:(x_k, y_k)]$ into Σ, while step 3 discards $L_j(x)$. In this case $[i:(x_i, y_i)]$ and $[k:(x_k, y_k)]$ become adjacent entries in Σ, indicating that the line segment from (x_i, y_i) to (x_k, y_k) is a part of the upper envelope. It is easy to show that the algorithm runs in $O(n)$ time. ■

For the base scenario, s_0 (defined in Section III), let $F_r^{s_0}(x)$ denote the upper envelope of the set of functions, $\{\underline{f}_i^r(x) \mid 1 \leq i \leq n\}$, where $\underline{f}_i^r(x) = \underline{f}_i(x)$ for x to the right of v_i, and 0 otherwise. A typical function $F_r^{s_0}(x)$ is plotted in

Figure 2. Symmetrically, let $F_l^{s_0}(x)$ denote the upper envelope

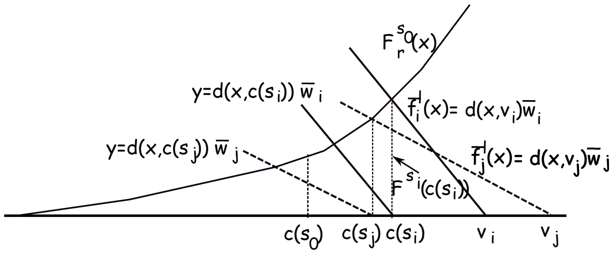
$F_r^{s_0}(x)$
$y=d(x,c(s_i))\,\overline{w}_i$
$\overline{f}_i^l(x)= d(x,v_i)\overline{w}_i$
$y=d(x,c(s_j))\,\overline{w}_j$
$\overline{f}_j^l(x)= d(x,v_j)\overline{w}_j$
$F^{s_i}(c(s_i))$
$c(s_0)$
$c(s_j)$
$c(s_i)$
v_i
v_j

Fig. 2. Intersection of $\overline{f}_i^l(x)$ and $F^{s_0}(x)$ gives $c(s_i)$.

of $\{\underline{f}_i^l(x) \mid 1 \le i \le n\}$, where $\underline{f}_i^l(x) = \underline{f}_i(x)$ for x to the left of v_i, and 0 otherwise. The value of $F_l^{s_0}(x)$ gets smaller as x approaches v_i from left. We have

$$F^{s_0}(x) = \max\{F_l^{s_0}(x), F_r^{s_0}(x)\},$$

and the 1-center under s_0, $c(s_0)$, is clearly at the lowest point of $F^{s_0}(x)$, namely at the intersection of $F_l^{s_0}(x)$ and $F_r^{s_0}(x)$. From Lemma 1 it follows that

Lemma 2: [4] The 1-center under s_0, $c(s_0)$, and $F_l^{s_0}(c(s_0)) = F_r^{s_0}(c(s_0))$ can be computed in $O(n)$ time. ■

B. Regret Function

Our next objective is to find $R^*(x)$. But to do so, we need to know *some* of the centers in $\{c(s_i) \mid i = 1, 2, \ldots, n\}$. Center $c(s_i)$ can be found by computing the intersection of $\overline{f}_i(x) = d(x, v_i)\overline{w}_i$ and $F^{s_0}(x)$. Recall that $F^{s_0}(x) = F_l^{s_0}(x)$ (resp. $= F_r^{s_0}(x)$) for x to the left (resp. right) of $c(s_0)$. Let $c(s_0)$ be on edge (v_m, v_{m+1}). Let $\overline{f}_i^l(x) = \overline{f}_i(x) = d(v_i, x)\overline{w}_i$ for x to the left of v_i, and 0 otherwise. Figure 2 shows the intersection of $\overline{f}_i^l(x)$ and $F_r^{s_0}(x)$, where $m+1 \le i \le n$. If $\overline{f}_i^l(x)$ passes below the lowest point of $F^{s_0}(x)$ at $x = c(s_0)$, then we have $c(s_i) = c(s_0)$. For $1 \le i \le m$, we can similarly compute $c(s_i)$, as the intersection of $\overline{f}_i^r(x)$ and $F_l^{s_0}(x)$. Again, if $\overline{f}_i^r(x)$ passes below the lowest point of $F^{s_0}(x)$ at $x = c(s_0)$, then we have $c(s_i) = c(s_0)$.

Function $R^*(x)$ will be convex, and the optimal location x^* will correspond to its lowest point. By definition, s_i is obtained from s_0 by changing w_i from $\underline{w}_i$ to $\overline{w}_i$. From (1), we have

$$\begin{aligned} F^{s_i}(x) &= \max_{v \in V} d(v, x) w_v^{s_i} \\ &= \max\{\overline{f}_i(x), F^{s_0}(x)\}. \end{aligned} \tag{7}$$

Cf. Lemma 3.5 in [6]. From (2), (4) and (7), we get

$$\begin{aligned} R^*(x) &= \max_{s_i \in \mathcal{S}^*} \{\max\{\overline{f}_i(x), F^{s_0}(x)\} - F^{s_i}(c(s_i))\} \\ &= \max_{s_i \in \mathcal{S}^*} \{\max\{d(c(s_i), x)\overline{w}_i, F^{s_0}(x) - F^{s_i}(c(s_i))\}\} \\ &= \max\{\max_{s_i \in \mathcal{S}^*} \{d(c(s_i), x)\overline{w}_i\}, F^{s_0}(x) - F^{s_k}(c(s_k))\}, \end{aligned} \tag{8}$$

where $F^{s_k}(c(s_k)) = \min_{s_i \in \mathcal{S}^*} F^{s_i}(c(s_i))$. Therefore, $R^*(x)$ can be computed as the upper envelope of $\{d(c(s_i), x)\overline{w}_i \mid s_i \in \mathcal{S}^*\}$ and $F^{s_0}(x) - F^{s_k}(c(s_k))$.

Figure 2 shows $F_r^{s_0}(x)$ and two cost functions $\overline{f}_i^l(x)$ and $\overline{f}_j^l(x)$, where $i < j$ and $\overline{w}_i > \overline{w}_j$. If $c(s_i)$ lies to the right of $c(s_j)$, then $\overline{f}_i^l(x) - F^{s_i}(c(s_i)) = d(c(s_i), x)\overline{w}_i$ dominates $\overline{f}_j^l(x) - F^{s_j}(c(s_j)) = d(c(s_i), x)\overline{w}_j$, and thus s_j can be discarded. In this case, $c(s_j)$ need not even be computed, saving time.

Algorithm `Find-Nondominated`
Push point $p = (c(s_0), F_r^{s_0}(c(s_0)))$ into stack $\mathcal{ND}$.
For $i = m+1, \ldots, n$, do the following:

1) Find if $\overline{f}_i^l(x)$ passes above point $p = (x_p, y_p)$ at the top of $\mathcal{ND}$.
2) If it does, compute the intersection of $\overline{f}_i^l(x)$ and $F_r^{s_0}(x)$, and push it into $\mathcal{ND}$. Otherwise, $c(s_i)$ is to the left of x_p, and $d(c(s_i), x)\overline{w}_i$ could not be a part of the upper envelope. In Figure 2, s_j satisfies this condition. ■

Lemma 3: Algorithm `Find-Nondominated` runs in $O(n)$ time.

Proof: By Lemma 2, we can compute $(c(s_0), F_r^{s_0}(c(s_0)))$ in $O(n)$ time. Step 1) of the above algorithm entails evaluating $\overline{f}_i^l(x_p)$, and takes constant time. In Step 2), we check successive segments of $F_r^{s_0}(x)$ upwards, starting at $p = (x_p, y_p)$. We never backtrack along $F_r^{s_0}(x)$, so that the total time required is $O(n)$. ■

Let us rewrite $\{d(c(s_i), x)\overline{w}_i \mid s_i \in \mathcal{S}^*\}$ in (8) as follows:

$$\begin{aligned} \{d(c(s_i), x)\overline{w}_i \mid s_i \in \mathcal{S}^*\} &= \{d(c(s_i), x)\overline{w}_i \mid 1 \le i \le m\} \\ &\quad \cup \{d(c(s_i), x)\overline{w}_i \mid m+1 \le i \le n\} \end{aligned} \tag{9}$$

In order to find the upper envelope of the functions in $\{d(c(s_i), x)\overline{w}_i \mid m+1 \le i \le n\}$ and identify its linear sections, we use Algorithm `Find-Envelope/Path`, with left and right reversed, starting at the right end of the path. This envelope is a continuous, piecewise linear, decreasing function of x, if we move x from v_1 towards v_n. We combine it with $F^{s_0}(x) - F^{s_k}(c(s_k))$, according to (8), to find the left half of $R^*(x)$. Analogously, we can find the right half of $R^*(x)$, based on $\{d(c(s_i), x)\overline{w}_i \mid 1 \le i \le m\}$, which is a continuous, piecewise linear, increasing function of x. The lowest point of $R^*(x)$ can be found in $O(n)$ time.

Theorem 2: The minmax regret 1-center of a path network can be computed in $O(n)$ time. ■

V. Cycle Network

Let $\mathcal{C} = (V, E)$ be a cycle with length $L_{\mathcal{C}}$. For $p, q \in \mathcal{C}$, the part of $\mathcal{C}$ clockwise, from p, to q is denoted by $\mathcal{C}(p, q)$, and its length is denoted by $d(p, q)$. The *cost* of a point $x \in \mathcal{C}$ under s is given by

$$F^s(x) = \sum_{v \in V} \min\{d(v, x), d(x, v)\} w_v^s. \tag{10}$$

Suppose that the vertices of $\mathcal{C}$ are numbered cw as $v_1, \ldots, v_n$ from the *origin* o that lies in the interior of edge (v_n, v_1). If $d(p, q) = d(q, p)$, we say that points p and q are *antipodal* to each other, and p (resp. q) is the *antipode* [7] of q (resp. p), denoted by $p = \alpha(q)$ (resp. $q = \alpha(p)$).

Recall the base scenario s_0 and the set $\mathcal{S}^*$ of single-max scenarios from Section III. To evaluate (2) for different scenarios and points, our first task is to find the centers $\{c(s_i) \,|\, s_i \in \mathcal{S}^*\}$.

A. *Finding Upper Envelope*

The cost line of vertex v_i under base scenario s_0, is $\underline{f}_i(x) = d(x, v_i)\underline{w}_i$. Figure 3 shows $\underline{f}_2(x)$, $\underline{f}_3(x)$, and $\underline{f}_4(x)$. They are plotted over two periods of a cycle, so that for any point $p \in \mathcal{C}$, one of the two occurrences of p in it has the property that $\mathcal{C}(\alpha(p), p)$ and $\mathcal{C}(p, \alpha(p))$ appear continuously in the diagram. This property becomes useful when we process "queries" later. Observe that $\underline{f}_i(x)$ takes the minimum (resp. maximum) value

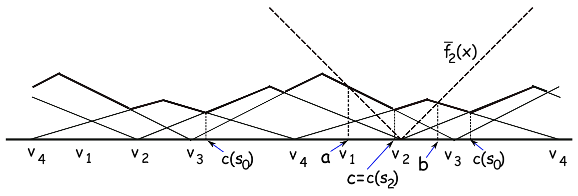

Fig. 3. The upper envelope is shown in thick line segments.

at $x = v_i$ (resp. $x = \alpha(v_i)$). The center $c(s_0)$ is at the lowest point of the upper envelope of $\{\underline{f}_i(x) \,|\, i = 1, 2, \ldots, n\}$.

The following algorithm constructs the *clockwise (cw) upper envelope*, $\mathcal{E}_{cw}$, of $\{L_i(x) \mid i = 1, 2, \ldots, n\}$, where $L_i(x) = \underline{f}_i(x)$ for $x \in \mathcal{C}(v_i, \alpha(v_i))$ and $L_i(x) = 0$ for $x \in \mathcal{C}(\alpha(v_i), v_i)$. It is very similar to `Find-Envelope/Path`.

Algorithm `Find-Envelope/Cycle`

```
1) Put $[1 : (v_1, 0)]$ in stack $\Sigma$. (Meaning: $L_1(x)$, starting at
   $(v_1, 0)$ is the initial part of the upper envelope.)
2) For $k = 2, 3, \ldots, n$, carry out steps 3 and 4.
3) Let $[i : (x_i, y_i)]$ be the item at the top of $\Sigma$. If line $L_k(x)$
   passes above $(x_i, y_i)$ pop up the top item from $\Sigma$ and
   repeat this step.
4) Compute the intersection $(x_k, y_k)$ of $L_k(x)$ and $L_i(x)$.
   If the intersection lies within $L_{\mathcal{C}}/2$ clockwise from $v_i$,
   then push $[k : (x_k, y_k)]$ into $\Sigma$.  ■
```

Clearly, there are $O(n)$ linear segments in $\mathcal{E}_{cw}$. The following lemma can be proved similarly to Lemma 1.

Lemma 4: When `Find-Envelope/Cycle` applied to a cycle completes, an entry $[i : (x_i, y_i)]$ in stack Σ means $L_i(x)$ forms a segment of the upper envelope of $\{L_j(x) \mid j = 1, 2, \ldots, n\}$, starting at point (x_i, y_i). `Find-Envelope/Cycle` runs in $O(n)$ time. ■

Similarly, we can compute the *counterclockwise (ccw) upper envelope*, $\mathcal{E}_{ccw}$, in $O(n)$ time.

Lemma 5: The 1-center, $c(s_0)$, of the base scenario on a cycle network can be found in $O(n)$ time.

Proof: After computing $\mathcal{E}_{cw}$ and $\mathcal{E}_{ccw}$ in $O(n)$ time, we compute their upper envelope, $\mathcal{E}$, in linear time. The lowest point of the merged envelope corresponds to $c(s_0)$. ■

For each single-max scenario $s_j \in \mathcal{S}^*$, we consider $\overline{f}_j(x)$ as its *query function*. In the example shown in Figure 3, the query function $\overline{f}_2(x)$ intersects upper envelope $\mathcal{E}$ at two points, a and b. Point c is the lowest point in $\mathcal{C}(a, b)$, so $c(s_2)$ is given by point c. In general, let a (resp. b) be the closest point to v_j, if any, where query line $\overline{f}_j(x)$ intersects $\mathcal{E}$ in the one-period interval $\mathcal{C}(\alpha(v_j), \alpha(v_j))$. If there is no such intersection point, set $a = \alpha(v_j)$ that lies on the ccw side of v_j in the 2-period diagram of $\mathcal{E}$, and $b = \alpha(v_j)$ that lies on the cw side of v_j in the 2-period diagram of $\mathcal{E}$. Then $c(s_j)$ is given by the lowest point on $\mathcal{E}$, clockwise from a to b.

Lemma 6: The centers $\{c(s_i) \,|\, s_i \in \mathcal{S}^*\}$ and the costs of the centers $\{F^{s_i}(c(s_i)) \,|\, s_i \in \mathcal{S}^*\}$ can be computed in $O(n \log n)$ time.

Proof: The intersection points a and b mentioned above, if any, can be determined in $O(\log n)$ time, using the query algorithm in [8] twice. It yields both $c(s_i)$ and $F^{s_i}(c(s_i))$. We repeat this for every scenario $s_i \in \mathcal{S}^*$. ■

B. *Finding the Optimal Location*

Let G be a network, which is a cycle in our current situation. Averbakh and Berman [9] convert G to its *auxiliary network* G' in such a way that the minmax regret 1-center problem on G becomes the classical 1-center problem on G'. Assume that $F^s(c(s))$ for all $s \in \mathcal{S}^*$ are available, and let M be a positive integer that is sufficiently large [9]. They append an edge (v_i, v_i') of length $\{M - F^{s_i}(c(s_i))\}/\overline{w}_i$ to every vertex $v_i \in V$, where v_i' is a new vertex of degree 1, called the *dummy vertex* corresponding to v_i. The vertices of G' that are present in G have weights equal to zero. For any vertex $v_i' \in G'$ its weight is set to $\overline{w}_i$. Clearly, G' is a weighted cactus graph. Averbakh and Berman prove

Theorem 3: [9] The minmax regret 1-center problem on network G can be solved by computing the classical weighted 1-center problem on G'. ■

Now that we have computed $\{F^{s_i}(c(s_i)) \mid s_i \in \mathcal{S}^*\}$ (Lemma 6), we can invoke Theorem 3. It is known that the classical weighted 1-center problem on a cactus network can be computed in $O(n \log n)$ time [10]. We thus have

Theorem 4: The minmax regret 1-center in a cycle network can be found in $O(n \log n)$ time. ■

VI. Tree Network

Since our algorithm for a tree network needs a balanced tree, we first review *spine decomposition* [11], [12] that can convert any tree into a structure that has properties of a balanced tree.

A. *Review of Spine Decomposition*

Many efficient tree algorithms preprocess a given tree, by aggregating information on its subtrees, and storing the condensed information at their root vertices. In *spine decomposition*, the structure that stores the aggregate information is separated from the given tree, and the height of this structure is bounded by $O(\log n)$, so that it can provide benefits of a balanced tree.

Given tree T with n vertices, we first find its 1-center under the base scenario $r_T = c(s_0)$, and assume that T is a binary tree rooted at $r_T = c(s_0)$. If T is not binary, it can be transformed into a binary tree by adding no more than

n zero weight vertices and zero length edges [13]. In spine decomposition, we select a path from r_T to a leaf in T such that the next vertex on this path always follows the child vertex that leads to the largest number of leaves from it. More formally, let $N_l(v)$ denote the number of leaves that have v as an ancestor. If $v_0(=r_T), v_1, \ldots, v_k$ are the vertices on the path, then $N_l(v_{i+1}) \geq N_l(v'_i)$ always holds, where v'_i is the other child vertex of v_i, if any. We call path $\langle v_0, v_1, \ldots, v_k\rangle$ the *top spine*, if $k \geq 1$. Now, for each $i = 1, \ldots, k$, we consider v_i as the root of the subtree containing v'_i, and find other *spines* recursively. Note that a vertex belongs to at most two spines. Between these two spines, the one that is closer to the root of T is considered the *parent spine* of the other. For the tree T in Figure 4(a), its spine decomposition is shown in Figure 4(b), where the vertices of each spine are aligned either horizontally or vertically.

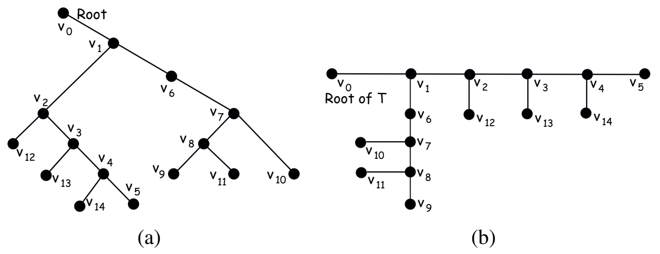

Fig. 4. (a) Tree T; (b) Spine decomposition of T.

A spine could be of length $O(n)$. To gather information efficiently from the vertices of spine σ, we build an auxiliary binary tree, called the *spine tree* and denoted by $\tau(\sigma)$, on top of it as follows. For spine $\sigma = \langle v_0, v_1, \ldots, v_k\rangle$ with root $r(\sigma) = v_0$, let $N(v_i)$ denote the number of descendant vertices of v_i in T, excluding the vertices on σ, but including v_i itself. Therefore, we have $N(v_0) = 1$. Under the root of $\tau(\sigma)$, we create two child nodes. (We use the term "nodes" to distinguish them from the vertices of the original tree T.) We partition $\{v_0, \ldots, v_k\}$ into two subsets $L = \{v_1, \ldots, v_h\}$ and $R = \{v_{h+1}, \ldots, v_k\}$ in such a way that $\sum_{i=0}^{h} N(v_i)$ and $\sum_{i=h+1}^{k} N(v_i)$ are as close as possible, and associate L (resp. R) with the left (resp. right) child node. We keep partitioning until each node is associated with just one vertex, which becomes a leaf node of $\tau(\sigma)$. We then identify the root of $\tau(\sigma)$ with $r(\sigma)$ in the parent spine. In Figure 5(a), for example, the nodes at the two ends of each arrow are really just one node. The resulting structure, together with the spine tree for each spine, is denoted by $SD(T)$.

Lemma 7: [12] $SD(T)$ can be constructed in $O(n \log n)$ time, where n is the number of vertices in the given tree T. ■

B. Envelope Tree

We remove the edges belonging to the original tree T from $SD(T)$ as in Figure 5(b), and call the resulting tree the *envelope tree*, denoted by $\mathcal{E}(T)$. An argument in [12] (Subsection 2.2) directly implies that

Lemma 8: [12] The height of $\mathcal{E}(T)$ is $O(\log n)$, where n is the number of vertices in T. ■

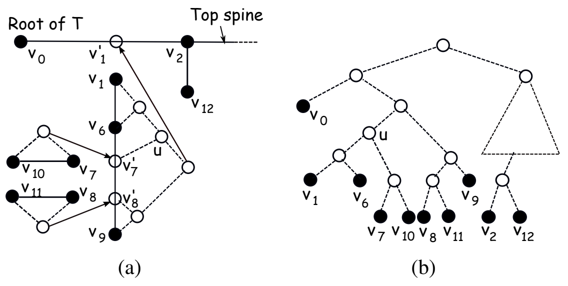

Fig. 5. (a) Part of $SD(T)$; (b) $\mathcal{E}(T)$.

Note that Lemma 8 does not imply that the sum of the heights of all spine trees is $O(\log n)$. For our purpose, we slightly modify spine decomposition. *As the top spine, we choose a maximal path from the root that contains the center* $c(s_0)$. Lemma 8 is still valid with this modification.

C. Algorithm

For a path and a cycle, instead of computing the intersection of $\overline{f}_j(x)$ with $\underline{f}_i(x)$ individually for all i, which is too time-consuming, we constructed the upper envelope for $\{\underline{f}_i(x) \mid v_i \in V\}$, and computed the intersection of $\overline{f}_j(x)$ with it. However, it is difficult to make this approach work for a tree.

For a tree, we first compute the upper envelopes for various subsets of $\{\underline{f}_i(x) \mid v_i \in V\}$ instead. Namely, in Phase 1, we compute an upper envelope at each node (called an *envelope node*) u in $\mathcal{E}(T)$. Such an upper envelope is for the cost functions of all vertices of T that have u as an ancestor in $\mathcal{E}(T)$, based on the vertex weights under s_0. For example, at u in Figure 5(b), we compute and store the upper envelope for its descendant vertices i.e., v_1, v_6, v_7 and v_{10}. In Phase 2, for each vertex $v \in V$, we extract a set of $O(\log n)$ upper envelopes from those computed in Phase 1, compute the intersection of $\overline{f}_j(x)$ with each of them, and determine the most costly one among them, which gives $c(s_j)$. Here are the details.

1) Phase 1: We assume that envelope tree $\mathcal{E}(T)$ has already been constructed for a given T (Lemma 7). At a leaf node of $\mathcal{E}(T)$, which is a vertex (v_i) of T, the upper envelope is $\underline{f}_i(x) = d(x, v_i)\underline{w}_i$, which is composed of either one line (if v_i is a leaf vertex of T) or two lines (if v_i is a non-leaf vertex of T) forming the shape of letter V. At the 2nd level from the bottom of $\mathcal{E}(T)$, each envelope node has two end vertices of an edge of T as its descendants. Eventually, we reach an envelope node that is the root of a spine tree. Figure 6 illustrates how to propagate the cost contributions from the vertices of a lowest spine σ_1 to its parent spine σ_2. The horizontal line in Figure 6(a) represents σ_1. It is connected to σ_2 at its root vertex $v = r(\sigma_1)$. The information concerning the upper envelope from σ_1 that we need to propagate to $\tau(\sigma_2)$ is shown by the thick line segments, labeled "Contribution to parent spine" in Figure 6(a). It, together with its mirror image, is stored as envelope $E_v(x)$ at v. This is because the cost contribution from a vertex on σ_1 is the same at any two points on σ_2 that are at equal distance from v, one closer to $r(\sigma_2)$ and the other away from it. See Figure 6(b).

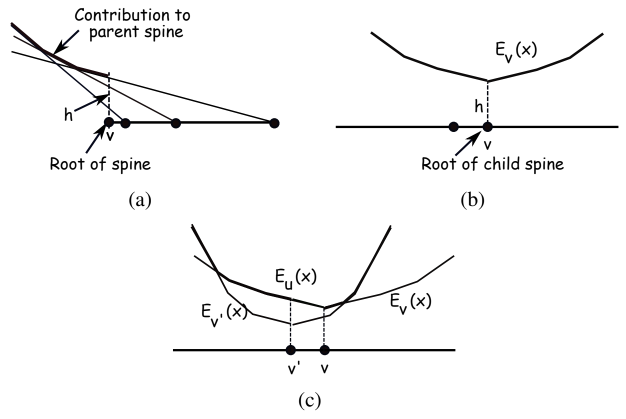

Fig. 6. (a) Spine σ_1 with $v=r(\sigma_1)$; (b) $E_v(x)$ stored at v on spine σ_2; (c) $E_u(x)$ stored at u (parent of v and v').

Suppose that two vertices on σ_2, v and v', have an envelope node u as their parent node. We now construct $E_u(x)$ from $E_v(x)$ and $E_{v'}(x)$ by computing their upper envelope. In Figure 6(c), $E_u(x)$ is shown by thick line segments. We can similarly construct the upper envelopes at higher envelope nodes in $\mathcal{E}(T)$. Since $\mathcal{E}(T)$ is of height $O(\log n)$ (Lemma 8), the total time required for computing the envelopes at all the nodes of $\mathcal{E}(T)$ is $O(n \log n)$.

2) Phase 2: Let $\pi(v_j)$ denote the path from the leaf node v_j (a vertex of T) to the root of $\mathcal{E}(T)$. We compute $c(s_j)$, tracing $\pi(v_j)$ upward from v_j as follows:

Algorithm `Find-Center/Tree`

```
1) At each node $u$ visited, do the following:
    (a) If the left (resp. right) child node of $u$ is on $\pi(v_j)$,
        let $u'$ denote $u$'s right (resp. left) child node.
    (b) Find the lowest intersection point, if any, of $\overline{f}_j(x)$
        with $E_{u'}(x)$, where they have opposite slopes.
2) From among the intersection points computed in step
   1(b), find the highest point. If this point has a higher
   cost than $c(s_0)$, then the corresponding location gives
   $c(s_j)$. Otherwise $c(s_j)=c(s_0)$. ■
```

Since $c(s_0)$ is on the top spine, if v_j is on spine σ, $c(s_j)$ can lie either on σ or an anscester spine. Thus, the spine trees for the other spines can be ignored. Suppose, for example, that we want to compute $c(s_{10})$ in Figure 5. We first visit the parent node of v_{10}, and check its other child node, where the upper envelope (i.e., $\underline{f}_7(x)$) for v_7 is stored as $E_{v_7}(x)$. We now carry out step 1(b), i.e., compute the intersection of $\overline{f}_{10}(x)$ and $\underline{f}_7(x)$, to find the first candidate for $c(s_{10})$. We then trace $\mathcal{E}(T)$ upwards to u, and repeat step 1(a)(b). In this case, it entails computing the intersection of $\overline{f}_{10}(x)$ with $E_{u'}(x)$ stored at the left child u' of u, where $E_{u'}(x)$ is the upper envelope of $\underline{f}_1(x)$ and $\underline{f}_6(x)$.

Theorem 5: The minmax regret 1-center of a tree network can be found in $O(n \log^2 n)$ time.

Proof: In executing `Find-Center/Tree`, the number of envelope nodes encountered on $\pi(v_j)$ is $O(\log n)$ by Lemma 8. Step 1(b) can be carried out, using binary search, since $E_{u'}(x)$ is convex with a number of corner points. Thus the processing time per vertex (v_j) is $O(\log^2 n)$, and the total time for all 1-centers $\{c(s_j) \mid s_i \in \mathcal{S}^*\}$ is $O(n \log^2 n)$. Once we have computed $\{c(s_i) \mid s_i \in \mathcal{S}^*\}$, we can evaluate $\{F^{s_i}(c(s_i)) \mid s_i \in \mathcal{S}^*\}$ easily, and invoke Theorem 3. ■

VII. Conclusion and Future Work

We have presented an algorithm for finding the minmax regret 1-center of a path (resp. cycle) network that runs in $O(n)$ (resp. $O(n \log n)$) time. No algorithms specifically tailored to these families of networks were previously known. We also presented a new algorithm for a tree that runs in $O(n \log^2 n)$ time. For a tree, an algorithm with the same complexity was already known [6], but its description takes up several pages.

Lemma 5 implies that the classical weighted 1-center in a cycle network can be computed in $O(n)$ time, which is a new result and is optimal. It is valid for any cycle network that contains at least one vertex whose weight is non-negative. As future work, we want to see if the $O(n \log^2 n)$ time complexity for a tree can be improved, and also work on a cactus network. A more challenging problem is finding the minmax regret p-center for $p \geq 2$.

Acknowledgment

This work was partially supported by Discovery Grants from the Natural Science and Engineering Research Council of Canada.

References

[1] T. S. Hale and C. R. Moberg, "Location science research: A review," *Annals of Operations Research*, vol. 123, pp. 21–35, 2003.
[2] B. Chen and C.-S. Lin, "Minmax-regret robust 1-median location on a tree," *Networks*, vol. 31, pp. 93–103, 1998.
[3] O. Kariv and S. Hakimi, "An algorithmic approach to network location problems, part 1: The p-centers," *SIAM J. Appl. Math.*, vol. 37, pp. 513–538, 1979.
[4] N. Megiddo, "Linear-time algorithms for linear-programming in R^3 and related problems," *SIAM J. Computing*, vol. 12, pp. 759–776, 1983.
[5] I. Averbakh and O. Berman, "Algorithms for the robust 1-center problem on a tree," *European Journal of Operational Research*, vol. 123, no. 2, pp. 292–302, 2000.
[6] H.-I. Yu, T.-C. Lin, and B.-F. Wang, "Improved algorithms for the minmax-regret 1-center and 1-median problem," *ACM Transactions on Algorithms*, vol. 4, no. 3, pp. 1–1, June 2008.
[7] A. Goldman, "Optimal center location in simple networks," *Transportation Science*, vol. 5, pp. 212–221, 1971.
[8] B. Chazelle and L. J. Guibas, "Fractional cascading: II. Applications," *Algorithmica*, vol. 1, pp. 163–191, 1986.
[9] I. Averbakh and O. Berman, "Minimax regret p-center location on a network with demand uncertainty," *Location Science*, vol. 5, pp. 247–254, 1997.
[10] B. Ben-Moshe, B. Bhattacharya, Q. Shi, and A. Tamir, "Efficient algorithms for center problems in cactus networks," *Theoretical Compter Science*, vol. 378, no. 3, 2007.
[11] R. Benkoczi, B. Bhattacharya, M. Chrobak, L. Larmore, and W. Rytter, "Faster algorithms for k-median problems in trees," *Mathematical Foundations of Computer Science, Springer Verlag*, vol. LNCS 2747, pp. 218–227, 2003.
[12] R. Benkoczi, "Cardinality constrained facility location problems in trees," Ph.D. dissertation, School of Computing Science, Simon Fraser University, Canada, 2004.
[13] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and the related problems in tree graphs," *Operations Research Letters*, vol. 19, pp. 59–64, 1996.