European Journal of Operational Research 239 (2014) 636–643

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Minmax regret 1-facility location on uncertain path networks

Haitao Wang *

Department of Computer Science, Utah State University, Logan, UT 84322, USA

ARTICLE INFO

Article history:
Received 29 November 2013
Accepted 19 June 2014
Available online 28 June 2014

Keywords:
Algorithms
Path networks
Uncertainty
Facility location
Minmax regret

ABSTRACT

Let P be an undirected path graph of n vertices. Each edge of P has a positive length and a constant capacity. Every vertex has a nonnegative supply, which is an unknown value but is known to be in a given interval. The goal is to find a point on P to build a facility and move all vertex supplies to the facility such that the maximum regret is minimized. The previous best algorithm solves the problem in $O(n\log^2 n)$ time and $O(n\log n)$ space. In this paper, we present an $O(n\log n)$ time and $O(n)$ space algorithm, and our approach is based on new observations and algorithmic techniques.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Facility location problems on networks have received considerable attention over a few decades. The problems are normally concerned with networks where the information (e.g., the vertex and the edge weights) are known precisely. However, data in practice often involve uncertainty and may change with the time. Recently facility locations problems in uncertain environments have been studied, e.g., Averbakh and Bereg (2005), Averbakh and Berman (1997, 2000a, 2000b, 2003), Bhattacharya and Kameda (2012), Bhattacharya, Kameda, and Song (2012a, 2012b), Chen and Lin (1998), Cheng et al. (2013), Conde (2007, 2008), Kouvelis and Yu (1997), Puerto, Rodríguez-Chía, and Tamir (2009) and Yu, Lin, and Wang (2008). One approach that is often used to model the uncertainty is the *worst-case analysis* in which one is looking for a solution that performs reasonably well for all possible *scenarios* (where a scenario is a specific realization of all uncertain parameters of the problem). There are many optimization criteria in the worst-case analysis. In particular, the *minmax regret optimization* aims at obtaining a solution that minimizes the maximum deviation, over all possible scenarios, between the value of the solution and the optimal value of the corresponding scenario, e.g., Averbakh and Berman (1997, 2000b), Bhattacharya and Kameda (2012), Bhattacharya et al. (2012a, 2012b), Cheng et al. (2013), Kouvelis and Yu (1997) and Yu et al. (2008). In other words, the minmax regret optimization seeks to minimize the worst-case loss in the objective function value that may occur because the solution is chosen without knowing which scenario will take place.

In this paper, we consider the *minmax regret 1-facility location problem on uncertain path networks* where the vertex weights are uncertain. The problem was proposed recently by Cheng et al. (2013) and an $O(n\log^2 n)$ time and $O(n\log n)$ space algorithm was given in Cheng et al. (2013). By discovering more observations, we present an $O(n\log n)$ time and $O(n)$ space algorithm in this paper. Shortly after the preliminary version of this paper appeared in Wang (2013), Cheng et al.'s algorithm (Cheng et al., 2013) was independently improved to $O(n\log n)$ time and $O(n\log n)$ space in their journal paper (Higashikawa et al., 2014).

As discussed in Cheng et al. (2013), the problem is motivated by an earthquake evacuation problem due to the Tohoku-Pacific Ocean Earthquake that happened in Japan on March 11th, 2011. For example, suppose we have a highway that connects many cities and we want to find a location on the highway to build an evacuation facility such that when earthquake happens we can evacuate people in all these cities to the facility as soon as possible. The number of people in each city is uncertain due to different time periods (e.g., weekdays, weekends, days, nights, holidays). We formally introduce the problem below, and some notations are borrowed from Cheng et al. (2013).

1.1. Problem definitions

Let $P=(V,E)$ be a path graph, with the vertex set $V=\{v_1,\ldots,v_n\}$ and the edge set $E=\{e_1,\ldots,e_{n-1}\}$, such that e_i connects v_i and v_{i+1} for each $1\leqslant i\leqslant n-1$. Each edge $e\in E$ has a positive weight $l(e)$. Each vertex $v_i\in V$ has a weight w_i (e.g., the number of evacuees), which is unknown but is known in a given interval $[w_i^-,w_i^+]$ with $0\leqslant w_i^-\leqslant w_i^+$. Let c be a constant representing the *capacity* of each edge, which is the maximum number of

* Tel.: +1 435 797 2416; fax: +14357973265.
E-mail address: haitao.wang@usu.edu

http://dx.doi.org/10.1016/j.ejor.2014.06.026
0377-2217/© 2014 Elsevier B.V. All rights reserved.

evacuees passing any point in any unit time. Let τ be a positive constant representing the time required for traversing a unit distance of every evacuee. Let Σ be the Cartesian product of all intervals $[w_i^-, w_i^+]$ for $1 \leqslant i \leqslant n$. Every element $s \in \Sigma$ is called a *scenario* that is a feasible assignment of weights to the vertices of P. For any scenario $s \in \Sigma$, for each $1 \leqslant i \leqslant n$, we denote by $w_i(s)$ the weight of the vertex v_i in the scenario s, and $w_i^- \leqslant w_i(s) \leqslant w_i^+$.

As in Cheng et al. (2013), we embed the path P on a real line L (e.g., the x-axis) such that each vertex $v_i \in V$ is associated with the coordinate $x_i = x_1 + \sum_{j=1}^{i-1} l(e_j)$ for each $2 \leqslant i \leqslant n$. For each point $x \in L$, with a little abuse of notation, we also use x to denote the coordinate of the point. We use P to denote the set of points x on L with $x_1 \leqslant x \leqslant x_n$. For any point $x \in P$, let $P_L(x) = \{t \in P \mid t < x\}$ and $P_R(x) = \{t \in P \mid t > x\}$. Suppose we build a facility at a location $x \in P$. Consider any scenario $s \in \Sigma$. We use $T_L(x,s)$ to denote the minimum time for the evacuees on $P_L(x)$ to move to x; similarly, let $T_R(x,s)$ denote the minimum time for the evacuees on $P_R(x)$ to move to x. Note that if x is at a vertex $v_i \in V$, then we assume the evacuees at v_i can complete evacuation in no time. As discussed in Cheng et al. (2013), by Kamiyama, Katoh, and Takizawa (2006), $T_L(x,s)$ and $T_R(x,s)$ can be expressed as follows.

$$T_L(x,s) = \max_{v_i \in P_L(x)} \left\{ (x - x_i) \cdot \tau + \left\lceil \frac{1}{c} \cdot \sum_{j=1}^{i} w_j(s) \right\rceil - 1 \right\},$$

$$T_R(x,s) = \max_{v_i \in P_R(x)} \left\{ (x_i - x) \cdot \tau + \left\lceil \frac{1}{c} \cdot \sum_{j=i}^{n} w_j(s) \right\rceil - 1 \right\}.$$

Note that if $P_L(x) = \emptyset, T_L(x,s) = 0$, and if $P_R(x) = \emptyset, T_R(x,s) = 0$.

As in Cheng et al. (2013) and Higashikawa et al. (2014), in this paper we consider the case where $c = 1$. We should point out that Cheng et al. (2013) claimed that any algorithm works for $c = 1$ can also work for any other values of c (our preliminary version (Wang, 2013) cited their claim). However, it was found that the claim was not correct (Higashikawa et al., 2014), and therefore, their algorithms (Cheng et al., 2013; Higashikawa et al., 2014) only work for the case $c = 1$ and so does the algorithm in this paper.

But we can ignore the -1 from the above formulas when designing the algorithm. Hence, as in Cheng et al. (2013), we simply use the following definitions for $T_L(x,s)$ and $T_R(x,s)$.

$$T_L(x,s) = \max_{v_i \in P_L(x)} \left\{ (x - x_i) \cdot \tau + \sum_{j=1}^{i} w_j(s) \right\},$$

$$T_R(x,s) = \max_{v_i \in P_R(x)} \left\{ (x_i - x) \cdot \tau + \sum_{j=i}^{n} w_j(s) \right\}.$$

As in Cheng et al. (2013), for convenience of discussion, for each $1 \leqslant i \leqslant n$, we define a function $f_L^i(x,s)$ on $x > x_i$ and a function $f_R^i(s,x)$ on $x < x_i$ as follows.

$$f_L^i(x,s) = (x - x_i) \cdot \tau + \sum_{j=1}^{i} w_j(s),$$

$$f_R^i(x,s) = (x_i - x) \cdot \tau + \sum_{j=i}^{n} w_j(s).$$

Hence, we have $T_L(x,s) = \max_{v_i \in P_L(x)} f_L^i(x,s)$ and $T_R(x,s) = \max_{v_i \in P_R(x)} f_R^i(x,s)$.

Let $T(x,s)$ denote the minimum time for all evacuees on P to move to x. Thus, $T(x,s) = \max\{T_L(x,s), T_R(x,s)\}$. Denote by $x_{opt}(s)$ a point on P such that $T(x,s)$ is minimized when $x = x_{opt}(s)$, and one may consider $x_{opt}(s)$ as an *optimal location* for the scenario s. For any point x on L, let $R(x,s) = T(x,s) - T(x_{opt}(s), s)$, and we call $R(x,s)$ the *regret* of x in the scenario s. Intuitively, $R(x,s)$ is the regret (i.e., the opportunity loss) caused by choosing the location x instead of the optimal location $x_{opt}(s)$. Finally, the *maximum regret* of x is defined as $R_{max}(x) = \max_{s \in \Sigma} R(x,s)$. In other words, $R_{max}(x)$ is the worst-case opportunity loss for choosing the location x.

Our *minmax regret* problem is to choose a location x on L such that the maximum regret $R_{max}(x)$ is minimized, and the minimized $R_{max}(x)$ is called the *minmax regret*.

1.2. Our approach

In this paper we present an algorithm of $O(n \log n)$ time and $O(n)$ space for the problem, which improves the $O(n \log^2 n)$ time and $O(n \log n)$ space algorithm (Cheng et al., 2013).

Our algorithm makes use of the critical observation given in Cheng et al. (2013) that there are a set S of $2n$ scenarios such that for any point x on L, the "worst-case" scenario for $R_{max}(x)$ must be in S. This implies that instead of considering the infinitely many scenarios of Σ for computing $R_{max}(x)$, we only need to consider the scenarios in S. The algorithm has two main steps. The first step is to compute the optimal positions for all scenarios in S. An $O(n \log^2 n)$ time algorithm is given in Cheng et al. (2013) for the step. By finding new properties on the optimal solutions of these scenarios, we are able to compute all optimal solutions in $O(n \log n)$ time by an even simpler algorithm. The second step is to compute the minmax regret. This step also takes $O(n \log^2 n)$ time in Cheng et al. (2013). Our algorithm runs in $O(n \log n)$ time and $O(n)$ space. The high level scheme of our approach is binary search, whose efficiency hinges on solving the following sub-problem in linear time: Given any point x on L, compute the values $T_L(x,s)$ and $T_R(x,s)$ for all scenarios $s \in S$. A straightforward method can compute $T_L(x,s)$ and $T_R(x,s)$ in $O(n)$ time for each scenario s, and thus solves the sub-problem in $O(n^2)$ time. By discovering some interesting observations, we present an $O(n)$ time algorithm for the sub-problem. It should be noted that our algorithm itself is very simple but it is more challenging to observe the crucial properties behind the scene.

In the following, we discuss some basic observations in Section 2. In Section 3, we compute the optimal locations for all scenarios in S. Section 4 computes the minmax regret. Section 5 concludes the paper and discusses some possible future work.

2. Preliminaries

We discuss some observations that will be useful for our algorithm. Most of these observations have been discovered in Cheng et al. (2013) and we sketch them in this section for completeness of this paper.

Our goal is to find a location x to minimize the maximum regret $R_{max}(x) = \max_{s \in \Sigma} R(x,s)$. Consider any point x on P and any scenario $s \in \Sigma$. To compute $R(x,s)$, we need to known $x_{opt}(s)$ first. Recall that $x_{opt}(s)$ is the value of x such that $T(x,s) = \max\{T_L(x,s), T_R(x,s)\}$ is minimized when $x = x_{opt}(s)$. To determine $x_{opt}(s)$, we discuss some properties of $T_L(x,s)$ and $T_R(x,s)$.

Recall that $T_L(x,s) = \max_{v_i \in P_L(x)} f_L^i(x,s)$. For each $1 \leqslant i \leqslant n$, the function $f_L^i(x,s)$ defines in the plane an open half-line of slope τ with (but excluding) the (left) endpoint $(x_i, \sum_{j=1}^{i} w_j(s))$ (e.g., see Fig. 1). $T_L(x,s)$ is the upper envelope of the n half-lines defined by the functions $f_L^i(x,s)$ for $i = 1, \ldots, n$. Since $\tau > 0$, $T_L(x,s)$ is a strictly increasing function of x (e.g., see Fig. 1). Similarly, each $f_R^i(x,s)$ defines an open half-line of slope $-\tau$ with (but excluding) the (right) endpoint $(x_i, \sum_{j=i}^{n} w_j(s))$, and $T_R(x,s)$ is the corresponding upper envelope, which is strictly decreasing. Since $T(x,s) = \max\{T_L(x,s), T_R(x,s)\}$, $T(x,s)$ is a *unimodal function* of x in the sense that there exists a value x^* such that $T(x,s)$ is strictly decreasing on $(-\infty, x^*]$ and increasing on $[x^*, +\infty)$ (e.g., see Fig. 2). Note that the above x^* is $x_{opt}(s)$. These properties are already given in Cheng et al. (2013). We also have the following observation.

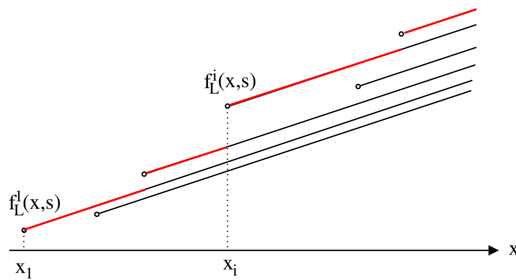

Fig. 1. Illustrating the functions $f_L^i(x,s)$. $T_L(x,s)$ is the upper envelope of them, shown with red color. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Observation 1. For any scenario $s \in \Sigma$, $T(x,s)$ is the upper envelope of the functions $f_L^i(x,s)$ and $f_R^i(x,s)$ for $i = 1,\ldots,n$.

For any point x, to compute the maximum regret $R_{max}(x)$, a straightforward approach is to enumerate all scenarios in Σ to compute $R(x,s)$ for every scenario $s \in \Sigma$. However, since there are infinitely many scenarios in Σ, the approach does not work. Below, we use a difference approach.

A scenario s is the *worst-case scenario* for the location x if $R_{max}(x) = R(x,s)$, and we denote it by $s(x)$. Clearly, if we know $s(x)$, then we can compute $R_{max}(x) = R(x,s(x))$. Cheng et al. (2013) provided a way to determine a set S of at most $2n$ scenarios such that $s(x)$ must be in S for any x, as follows.

For each $1 \leqslant i \leqslant n$, let s_L^i be the scenario where the weight $w_j(s_L^i)$ of the vertex v_j is w_j^+ for each j with $1 \leqslant j \leqslant i$, and $w_j(s_L^i) = w_j^-$ for each j with $i+1 \leqslant j \leqslant n$ if $i < n$. Symmetrically, for each $1 \leqslant i \leqslant n$, let s_R^i be the scenario where $w_j(s_R^i) = w_j^-$ for each j with $1 \leqslant j \leqslant i$, and $w_j(s_R^i) = w_j^+$ for each j with $i+1 \leqslant j \leqslant n$ if $i < n$. Let $S_L = \{s_L^i \mid 1 \leqslant i \leqslant n\}$ and $S_R = \{s_R^i \mid 1 \leqslant i \leqslant n\}$. Let $S = S_L \cup S_R$. The following lemma has been proved in Cheng et al. (2013).

Lemma 1 Cheng et al. (2013). *For any point x on L, there exists a worst-case scenario for x in S.*

In light of Lemma 1, we have $R_{max}(x) = \max_{s \in S} R(x,s)$. Hence, to compute $R_{max}(x)$, instead of considering all scenarios of Σ, we only need to consider the $2n$ scenarios in S. For each $s \in S$, to compute $R(x,s)$, we need to know the optimal location $x_{opt}(s)$. Cheng et al. (2013) presented an $O(n\log^2 n)$ time algorithm for computing $x_{opt}(s)$ for all scenarios $s \in S$, and in Section 3 we describe an $O(n\log n)$ time algorithm.

3. Computing the optimal solutions for the scenarios of *S*

In this section, we present an $O(n\log n)$ time and $O(n)$ space algorithm for computing $x_{opt}(s)$ for all scenarios $s \in S$, which improves the $O(n\log^2 n)$ time algorithm in Cheng et al. (2013).

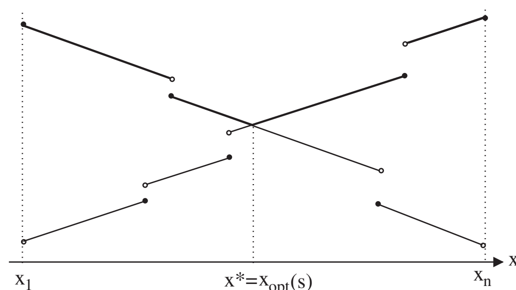

Fig. 2. Illustrating the function $T(x,s)$ shown with thick segments and the optimal location $x_{opt}(s)$.

Our improvement is due in a large part to certain monotonicity properties of the values $x_{opt}(s)$ given in Lemma 2.

Lemma 2. *For any two scenarios s_L^i and s_L^{i+1} of S_L with $1 \leqslant i \leqslant n-1$, if $x_{i+1} \leqslant x_{opt}(s_L^i)$, then $x_{i+1} \leqslant x_{opt}(s_L^{i+1}) \leqslant x_{opt}(s_L^i)$; otherwise, $x_{opt}(s_L^i) \leqslant x_{opt}(s_L^{i+1}) \leqslant x_{i+1}$.*

Proof. We only prove the case where $x_{i+1} \leqslant x_{opt}(s_L^i)$ since the proof for the other case where $x_{i+1} > x_{opt}(s_L^i)$ is very similar.

According to the definitions of the two scenarios s_L^i and s_L^{i+1}, for each vertex v_j, if $j \neq i+1$, the weights of v_j in the two scenarios are the same, but for the vertex v_{i+1}, $w_{i+1}(s_L^i) = w_{i+1}^-$ and $w_{i+1}(s_L^{i+1}) = w_{i+1}^+$. By Corollary 1 in Cheng et al. (2013), $x_{i+1} \leqslant x_{opt}(s_L^{i+1})$ holds. Below, we prove $x_{opt}(s_L^{i+1}) \leqslant x_{opt}(s_L^i)$. To this end, it is sufficient to show that $T_L(x,s_L^{i+1}) > T_R(x,s_L^{i+1})$ for any $x > x_{opt}(s_L^i)$. The details are given below.

Consider any value $x > x_{opt}(s_L^i)$. Since $T_L(x,s)$ is strictly increasing and $T_R(x,s)$ is strictly decreasing for any scenario s, according to the definition of $x_{opt}(s_L^i)$, we have $T_L(x,s_L^i) > T_R(x,s_L^i)$.

According to the definitions of s_L^i and s_L^{i+1}, $f_L^j(t,s_L^{i+1}) \geqslant f_L^j(t,s_L^i)$ for any $j \geqslant i+1$ and any $t > x_j$ (more precisely, $f_L^j(t,s_L^{i+1}) = f_L^j(t,s_L^i) + w_{i+1}^+ - w_{i+1}^-$), and $f_L^j(t,s_L^{i+1}) = f_L^j(t,s_L^i)$ for any $j \leqslant i$ and any $t > x_j$ (e.g., see Fig. 3). Due to $x > x_{opt}(s_L^i) \geqslant x_{i+1}$, we obtain $T_L(x,s_L^{i+1}) \geqslant T_L(x,s_L^i)$.

Similarly, $f_R^j(t,s_L^{i+1}) \geqslant f_R^j(t,s_L^i)$ for any $j \leqslant i+1$ and any $t < x_j$, and $f_R^j(t,s_L^{i+1}) = f_R^j(t,s_L^i)$ for any $j \geqslant i+2$ and any $t < x_j$ (e.g., see Fig. 3). Since $x > x_{opt}(s_L^i) \geqslant x_{i+1}$, none of the functions $f_R^j(t,s_L^{i+1})$ for $j \leqslant i+1$ is defined on $t = x$. Therefore, we obtain $T_R(x,s_L^{i+1}) = T_R(x,s_L^i)$.

The above shows that $T_L(x,s_L^i) > T_R(x,s_L^i)$, $T_L(x,s_L^{i+1}) \geqslant T_L(x,s_L^i)$, and $T_R(x,s_L^{i+1}) = T_R(x,s_L^i)$. Hence, we conclude that $T_L(x,s_L^{i+1}) > T_R(x,s_L^{i+1})$. □

Lemma 2 implies the following monotonicity property of $x_{opt}(s_L^i)$. Suppose initially $x_2 \leqslant x_{opt}(s_L^1)$; as the index i increases, $x_{opt}(s_L^i)$ moves monotonically "backward" to the left until at some moment $x_{i+1} > x_{opt}(s_L^i)$ happens, after which $x_{opt}(s_L^i)$ moves monotonically "forward" to the right. This monotonicity property turns out to be quite useful to our algorithm.

Similarly, we have the following lemma for S_R, which implies a monotonicity property of $x_{opt}(s_R^i)$ (the indices are considered from right to left).

Lemma 3. *For any two scenarios s_R^i and s_R^{i+1} of S_R with $1 \leqslant i \leqslant n-1$, if $x_i \leqslant x_{opt}(s_R^{i+1})$, then $x_i \geqslant x_{opt}(s_R^i) \geqslant x_{opt}(s_R^{i+1})$; otherwise, $x_{opt}(s_R^{i+1}) \geqslant x_{opt}(s_R^i) \geqslant x_i$.*

Proof. The proof is symmetric to that for Lemma 2 by considering the indices from right to left, and we omit the details. □

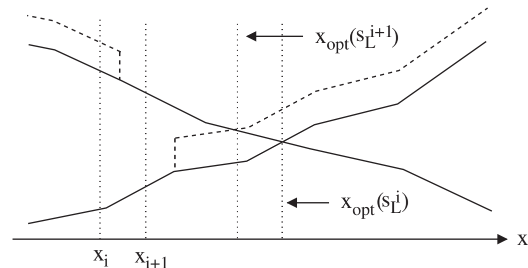

Fig. 3. Illustrating $T_L(x,s_L^i)$, $T_R(x,s_L^i)$, $T_L(x,s_L^{i+1})$, and $T_R(x,s_L^{i+1})$. $T_L(x,s_L^{i+1})$ (resp., $T_R(x,s_L^{i+1})$) can be obtained by shifting a portion of $T_L(x,s_L^i)$ (resp., $T_R(x,s_L^i)$) on the right (resp., left) of x_{i+1} upwards for $w_{i+1}^+ - w_{i+1}^-$.

Based on Lemmas 2 and 3, we present our algorithm for computing $x_{opt}(s)$ for all $s \in S$ as follows. We first compute $x_{opt}(s)$ for all $s \in S_L$, by using Lemma 2.

Our algorithm will compute $x_{opt}(s_L^i)$ in the index order $i = 1, 2, \ldots, n$. We assume we already have a data structure D that can compute the values $T_L(x, s)$ and $T_R(x, s)$ whenever needed for any x and $s \in S_L$. Initially, to determine $x_{opt}(s_L^1)$, we compute the values $T_L(x, s_L^1)$ and $T_R(x, s_L^1)$ for $x = x_1, x_2, \ldots$ in the (forward) order to find the smallest index i_1 such that $T_L(x_{i_1}, s_L^1) \geqslant T_R(x_{i_1}, s_L^1)$. As discussed in Cheng et al. (2013), $x_{opt}(s_L^1) \in [x_{i_1 - 1}, x_{i_1}]$ and can be determined in constant time. Next, we compute $x_{opt}(x_L^2)$. Assume $x_2 \leqslant x_{opt}(x_L^2)$. By Lemma 2, $x_2 \leqslant x_{opt}(x_L^2) \leqslant x_{opt}(x_L^1)$, we only need to search the portions of $T_L(x, s_L^2)$ and $T_R(x, s_L^2)$ for $x_2 \leqslant x \leqslant x_{opt}(s_L^1)$. To this end, we compute the values $T_L(x, s_L^2)$ and $T_R(x, s_L^2)$ by using D for $x = x_{i_1}, x_{i_1 - 1}, \ldots$ in the (backward) order to find the first index i_2 such that $T_L(x_{i_2}, s_L^2) \geqslant T_R(x_{i_2}, s_L^2)$ and $T_L(x_{i_2 - 1}, s_L^2) < T_R(x_{i_2 - 1}, s_L^2)$. As discussed in Cheng et al. (2013), $x_{opt}(s_L^2) \in [x_{i_2 - 1}, x_{i_2}]$ and can be determined in constant time.

In general, assume $x_{opt}\left(s_L^j\right)$ has been computed and $x_{j+1} \leqslant x_{opt}\left(s_L^j\right)$. Further, assume $x_{opt}\left(s_L^j\right)$ is known in the interval $[x_{i_j - 1}, x_{i_j}]$. To compute $x_{opt}\left(s_L^{j+1}\right)$, by Lemma 2, we have $x_{j+1} \leqslant x_{opt}\left(s_L^{j+1}\right) \leqslant x_{opt}\left(s_L^j\right)$. We compute the values $T_L\left(x, s_L^{j+1}\right)$ and $T_R\left(x, s_L^{j+1}\right)$ by D for $x = x_{i_j}, x_{i_j - 1}, \ldots$ in the (backward) order to find the first index i_{j+1} such that $T_L\left(x_{i_{j+1}}, s_L^{j+1}\right) \geqslant T_R\left(x_{i_{j+1}}, s_L^{j+1}\right)$ and $T_L(x_{i_{j+1} - 1}, s_L^{j+1}) < T_R(x_{i_{j+1} - 1}, s_L^{j+1})$. Again, $x_{opt}\left(s_L^{j+1}\right) \in [x_{i_{j+1} - 1}, x_{i_{j+1}}]$ and can be determined in constant time.

We continue the same procedure until the first time we have computed $x_{opt}(s_L^k)$ with $x_{opt}(s_L^k) < x_{k+1}$ for an index k. We also have the interval $[x_{i_k - 1}, x_{i_k}]$ that contains $x_{opt}(s_L^k)$. By Lemma 2, $x_{opt}(s_L^k) \leqslant x_{opt}(s_L^{k+1}) \leqslant x_{k+1}$. Hence, to compute $x_{opt}(s_L^{k+1})$, we need to search the portions of $T_L(x, s_L^{k+1})$ and $T_R(x, s_L^{k+1})$ for $x_{opt}(s_L^k) \leqslant x$. To this end, we compute the values $T_L(x, s_L^{k+1})$ and $T_R(x, s_L^{k+1})$ by D for $x = x_{i_k - 1}, x_{i_k}, \ldots$ in the (forward) order to find the first index i_{k+1} such that $T_L(x_{i_{k+1}}, s_L^{k+1}) \geqslant T_R(x_{i_{k+1}}, s_L^{k+1})$ and $T_L(x_{i_{k+1} - 1}, s_L^{k+1}) < T_R(x_{i_{k+1} - 1}, s_L^{k+1})$. Again, $x_{opt}(s_L^{k+1}) \in [x_{i_{k+1} - 1}, x_{i_{k+1}}]$ and can be determined in constant time. Next, we compute $x_{opt}(s_L^{k+2})$. We have the following observation.

Observation 2. $x_{opt}\left(s_L^{k+1}\right) < x_{k+2}$ holds.

Proof. By Lemma 2, we have $x_{opt}(s_L^k) \leqslant x_{opt}(s_L^{k+1}) \leqslant x_{k+1}$. Due to $x_{k+1} < x_{k+2}$, the observation simply follows. □

Due to the above observation, we can compute $x_{opt}(s_L^{k+2})$ in the similar way as $x_{opt}(s_L^{k+1})$. We continue this procedure to compute $x_{opt}(s_L^j)$ for $j = k+2, k+3, \ldots, n$. Note that similar observation as Observation 2 always holds (i.e., $x_{opt}(s_L^j) < x_{j+1}$ for any j with $k+1 \leqslant j \leqslant n-1$). The algorithm stops when $x_{opt}(s_L^n)$ is computed.

To analyze the running time, suppose any needed values $T_L(x, s)$ and $T_R(x, s)$ in the above algorithm can be computed in $O(T_D)$ time by using the data structure D; then we have the following lemma.

Lemma 4. *The values $x_{opt}(s)$ for all scenarios $s \in S_L$ can be computed in $O(n \cdot T_D)$ time.*

Proof. It is sufficient to show that the number of calls to D is $O(n)$ in the entire algorithm.

We still use k to denote the smallest index with $x_{opt}(s_L^k) < x_{k+1}$. By the monotonicity property in Lemma 2, $x_{opt}(s_L^i)$ is moving monotonically to the left for $i = 1, 2, \ldots, k$, and $x_{opt}(s_L^i)$ is moving monotonically to the right for $i = k+1, k+2, \ldots, n$. When we compute $x_{opt}(s_L^i)$ for $i = 1, \ldots, k$, the x values for computing $T_L(x, s_L^i)$ and $T_R(x, s_L^i)$ are monotone decreasing. Therefore, when computing the values $x_{opt}(s_L^i)$'s for $i = 1, \ldots, k$, the total number of calls on D is $O(n)$. Analogously, when computing the values $x_{opt}(s_L^i)$'s for $i = k+1, \ldots, n$, the total number of calls on D is also $O(n)$. The lemma thus follows. □

It remains to design the data structure D, which is given in the following lemma.

Lemma 5. *In $O(n)$ time and $O(n)$ space, we can build a data structure D that can compute in $O(\log n)$ time (i.e., $T_D = O(\log n)$) any value $T_L(x, s)$ or $T_R(x, s)$ needed in our algorithm for computing $x_{opt}(s)$ for all $s \in S_L$.*

Proof. With $O(n \log n)$ time preprocessing, Cheng et al. (2013) propose a data structure that can compute $T_L(x, s)$ and $T_R(x, s)$ for any x and $s \in S_L$ in $O(\log n)$ time, by using persistent data structures (Driscoll, Sarnak, Sleator, & Tarjan, 1989). Below, we give a simple solution with only $O(n)$ preprocessing time, without using the persistent data structures.

We first discuss an observation on our algorithm that makes the design of our data structure easier. In our algorithm for computing $x_{opt}(s)$ for all $s \in S_L$, when we are computing $x_{opt}(s_L^i)$, for any $1 \leqslant i \leqslant n$, we need to compute $T_L(x, s_L^i)$ and $T_R(x, s_L^i)$ for certain values of x. After $x_{opt}(s_L^i)$ is computed, we will never need to compute $T_L(x, s_L^i)$ and $T_R(x, s_L^i)$ for the scenario s_L^i again. Note that the corresponding algorithm in Cheng et al. (2013) does not have such a property.

Our data structure D has two parts D_L and D_R. D_L is for computing $T_L(x, s)$ and D_R is for computing $T_R(x, s)$. Below, we only discuss D_L since D_R is very similar.

D_L consists of a sequence of trees D_L^i for $i = 1, 2, \ldots, n$, where D_L^i is used for computing $T_L(x, s_L^i)$ for any x. Thanks to the observation discussed above, at any moment during the algorithm, we only need to maintain one tree in the above sequence (in contrast, because the corresponding algorithm in Cheng et al. (2013) does not have such a property, they have to maintain all these trees in a persistent data structure). Specifically, initially we construct the tree D_L^1. Then, for each $1 \leqslant i \leqslant n-1$, the tree D_L^{i+1} is obtained by updating the tree D_L^i in $O(\log n)$ time (D_L^i is thus destroyed). Below, we first describe the tree D_L^1 and then show how to update D_L^1 to obtain D_L^2. The tree is similar to that given in Cheng et al. (2013) (without being made persistent). We briefly discuss it here to make the paper self-contained.

We first discuss some observations on how to compute $T_L(x, s)$. Consider any scenario s and any value x with $x_{j-1} < x \leqslant x_j$ for certain j. Recall that the functions $f_L^1(x, s), f_L^2(x, s), \ldots, f_L^{j-1}(x, s)$ are defined on x while $f_L^j(x, s), f_L^{j+1}(x, s), \ldots, f_L^n(x, s)$ are not, and $T_L(x, s) = \max_{1 \leqslant t \leqslant j-1} f_L^t(x, s)$. Also recall that $f_L^t(x, s) = (x - x_t) \cdot \tau + \sum_{h=1}^{t} w_h(s)$. Hence, we can obtain the following $T_L(x, s) = x \cdot \tau + \max_{1 \leqslant t \leqslant j-1} (\sum_{h=1}^{t} w_h(s) - x_t \cdot \tau)$.

D_L^1 is a balanced binary search tree in which the leaves of D_L from left to right store the values $\sum_{h=1}^{t} w_h(s_L^1) - x_t \cdot \tau$ for $t = 1, 2, \ldots, n$. For each $1 \leqslant t \leqslant n$, let $a_t = \sum_{h=1}^{t} w_h(s_L^1) - x_t \cdot \tau$, which is stored in the t-th leaf. For each node v (either a leaf or an internal node), it also stores a value $max(v)$, which is equal to the maximum value stored in the leaves of the subtree rooted at v. The tree D_L^1 can be easily constructed in $O(n)$ time in a bottom-up manner. Given any value $x = x_j, T_L(x, s_L^1)$ can be computed in $O(\log n)$ time, as follows. According to our above discussion, we have $T_L(x_j, s_L^1) = x_j \cdot \tau + \max_{1 \leqslant t \leqslant j-1} a_t$. With standard techniques, by

following the path P_j in D_L^1 from the root to the j-th leaf, we find a minimum set V of nodes whose subtrees contain the leaves exactly from the first leaf to the $(j-1)$-th one. Specifically, $V = \{v \mid v \notin P_j$ and v is the left child of a node in $P_j\}$ (e.g., see Fig. 4). Clearly, $|V| = O(\log n)$ and can be found in $O(\log n)$ time. An easy observation is that the largest value $max(v)$ among all nodes $v \in V$ is exactly $\max_{1 \leqslant t \leqslant j-1} a_t$. Hence, $T_L(x_j, s_L^1)$ is equal to $x_j \cdot \tau$ plus the above largest value $max(v)$ of $v \in V$. Thus, we can compute $T_L(x_j, s_L^1)$ in $O(\log n)$ time.

Next, we show how to update D_L^1 in $O(\log n)$ time to obtain the tree D_L^2, which is for computing $T_L(x, s_L^2)$ for the scenario s_L^2. According to the definitions of the two scenarios s_L^1 and s_L^2, comparing with s_L^1, the weight of the vertex v_2 in s_L^2 increases by $w_2^+ - w_2^-$ while the weights of all other vertices are the same. Hence, all the values $\sum_{h=1}^t w_h(s_L^1) - x_t \cdot \tau$ for $t = 2, 3, \ldots, n$ stored in the leaves of D_L^1 except the leftmost leaf should increase by $w_2^+ - w_2^-$. We cannot afford to change each of these values explicitly since that would need $\Omega(n)$ time. To obtain an $O(\log n)$ time performance, we use the following approach. For each node v in the tree, we maintain an additional value, called the *supplement value* and denoted by $sup(v)$. In D_L^1, $sup(v) = 0$ for each node v. Hence, in D_L^1, for any $1 \leqslant t \leqslant n$, it holds that $a_t + \sum_{v \in P_t} sup(v) = \sum_{h=1}^t w_h(s_L^1) - x_t \cdot \tau$, where P_t is the path from the root of D_L^1 to the t-th leaf.

We update D_L^1 to obtain D_L^2 in the following way. Let P_2 be the path from the root of D_L^1 to the second leaf. First, for the second leaf v, we increase $sup(v)$ by $w_2^+ - w_2^-$. Then, for each node v that is not in P_2 but is a right child of a node of P_2, we increase $sup(v)$ by $w_2^+ - w_2^-$. Note that the above can be done in $O(\log n)$ time. Now consider the t-th leaf of the new tree, for any $1 \leqslant t \leqslant n$, and let P_t be the path from the root to the leaf. It is easy to see that $a_t + \sum_{v \in P_t} sup(v) = \sum_{h=1}^t w_h(s_L^2) - x_t \cdot \tau$. In other words, if we follow P_t from the root to aggregate the supplement value $sup(v)$, once we arrive the t-th leaf, we have the value $\sum_{h=1}^t w_h(s_L^2) - x_t \cdot \tau$ ready. Next, we update the values $max(v)$ for certain nodes v as follows. If a node v has a child whose supplement value has been increased above (note that v is necessarily on P_2), then its $max(v)$ may also need to be updated. To this end, for each internal node $v \in P_2$, we simply set $max(v)$ to be $\max\{max(u) + sup(u), max(w) + sup(w)\}$, where u and w are the two children of v. Note that we do not need to update the max value of the second leaf. This finishes our update on D_L^1 and the new tree is D_L^2. Clearly, D_L^2 can be obtained in $O(\log n)$ time.

Consider any internal node u on D_L^2 and suppose the leftmost (resp., rightmost) leaf in the subtree rooted at v is the l-th (resp., r-th) leaf. Let P_u be the path from the root to the node u. Based on our construction, $\max\{\sum_{h=1}^t w_h(s_L^2) - x_t \cdot \tau \mid l \leqslant t \leqslant r\}$ is exactly equal to the value $max(u) + \sum_{v \in P_u} sup(v)$. Hence, given any $x = x_j$, by using D_L^2, we can compute the value $T_L(x, s_L^2)$ in a similar way as before, and the only difference is that we need to aggregate the supplement values $sup(v)$ during traversing the tree from the root.

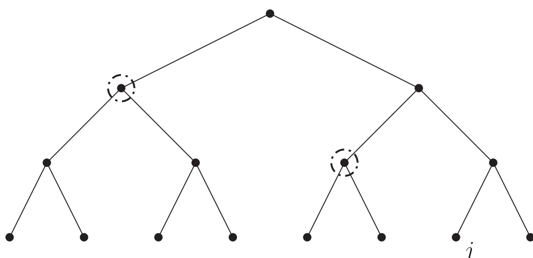

Fig. 4. The set V consists of the two circled nodes.

Specifically, to compute $T_L(x, s_L^2)$ for any $x = x_j$, let P_j be the path of D_L^2 from the root to the j-th leaf. We start from the root and traverse the path P_j to the j-th leaf. During the traversal, consider any node $v \in P_j$. We maintain a value $A(v)$, which is equal to $sup(v)$ if v is the root and $A(v.parent) + sup(v)$ otherwise (where $v.parent$ is the parent of v in P_j). If v has a left child u that is not in P_j, we let the value $max(u) + sup(u) + A(v)$ be in a set M ($M = \emptyset$ initially). If v is the j-th leaf, then we put the value $A(v) + max(v)$ in M and finish the traversal. After the traversal, M has $O(\log n)$ values, and the maximum value of M is equal to $\max\{\sum_{h=1}^t w_h(s_L^2) - x_t \cdot \tau \mid 1 \leqslant t \leqslant j-1\}$. Therefore, $T_L(x_j, s_L^2)$ is equal to the maximum value of M plus $x_j \cdot \tau$. Hence, $T_L(x_j, s_L^2)$ can be computed in $O(\log n)$ time.

Similarly, we can obtain the tree D_L^3 by updating D_L^2 in $O(\log n)$ time. In general, for any $1 \leqslant i \leqslant n-1$, if we already have the tree D_L^i, we can obtain D_L^{i+1} in $O(\log n)$ time by updating D_L^i such that we can compute $T_L(x, s_L^{i+1})$ for any $x = x_j$ in $O(\log n)$ time.

The lemma thus follows. □

Combining Lemmas 4 and 5, the values $x_{opt}(s)$ for all scenarios $s \in S_L$ can be computed in $O(n \log n)$ time. Using the similar algorithm and Lemma 3, we can also compute the values $x_{opt}(s)$ for all scenarios $s \in S_R$ in $O(n \log n)$ time. We conclude this section with the following theorem.

Theorem 1. *The values $x_{opt}(s)$ and $T(x_{opt}(s), s)$ for all scenarios $s \in S = S_L \cup S_R$ can be computed in $O(n \log n)$ time and $O(n)$ space.*

4. Computing the minmax regret

Our goal is to determine an *optimal location* x^* such that $R_{max}(x) = \max_{s \in \Sigma} R(x, s)$ is minimized at $x = x^*$, where $R(x, s) = T(x, s) - T(x_{opt}(s), s)$. Again, by Lemma 1, $R_{max}(x) = \max_{s \in S} R(x, s)$, which also implies that $R_{max}(x)$ is the upper envelope of the functions $R(x, s)$ for all $s \in S$.

Consider any scenario s. Since $T(x_{opt}(s), s)$ is a constant value and $T(x, s)$ is a unimodal function, $R(x, s)$ is also a unimodal function. Therefore, $R_{max}(x)$ is the upper envelope of a set of unimodal functions, which is also unimodal. To determine an optimal solution x^*, it is sufficient to determine the lowest point of the unimodal function $R_{max}(x)$. Due to the unimodality of $R_{max}(x)$, we will use binary search to find its lowest point.

The high-level scheme of our algorithm for finding x^* is a binary search on the values $x_1, x_2, \ldots, x_n$. For each value x_k considered in the binary search, we compute the value $R_{max}(x_k)$. To this end, we present an $O(n)$ time algorithm in Section 4.1 that can compute the values $T_L(x', s)$ and $T_R(x', s)$ for all $s \in S$, for any x', after which we can determine the value $R_{max}(x')$ in additional $O(n)$ time since we already know the values $T(x_{opt}(s), s)$ for all $s \in S$ by Theorem 1. Based on the function that gives the value $R_{max}(x_k)$, we can also determine which direction to do binary search in a standard way (Megiddo, 1983, 1984). The binary search will end up with either $x^* = x_i$ for some x_i or an interval (x_i, x_{i+1}) such that $x^* \in (x_i, x_{i+1})$. In the latter case, we finally determine x^* in additional $O(n)$ time by linear programming (Megiddo, 1983, 1984) as follows. Note that for any scenario s, the value $T_L(x, s)$ for $x \in (x_i, x_{i+1})$ are given by the same function $f_L^j(x, s)$ for some j, and similar observation holds for $T_R(x, s)$. We find the functions giving the values in the interval (x_i, x_{i+1}) for $T_L(x, s_L^i), T_R(x, s_L^i), T_L(x, s_R^i)$, and $T_R(x, s_R^i)$, for $i = 1, \ldots, n$. This can be done in $O(n)$ time by the same algorithm in Section 4.1. Denote by F the $O(n)$ functions computed above. Hence, x^* is the x-coordinate of the lowest point p^* of the upper envelope of the functions in F. Note that every function of F defines a half-line that spans the interval (x_i, x_{i+1}). Hence, although each function of F is a half-line, p^* is also the lowest point of the upper envelope of the

lines that contain the half-lines of F, and thus p^* can be computed in $O(n)$ time by linear programming (Megiddo, 1983, 1984).

4.1. A linear time algorithm for computing $T(x', s)$ for all $s \in S$

In this section, we present an $O(n)$ time algorithm for computing $T(x', s)$ for all $s \in S$, for any x'. In other words, our goal is to compute the values $T_L(x', s_L^i), T_R(x', s_L^i), T_L(x', s_R^i)$, and $T_R(x', s_R^i)$, for $i = 1, \ldots, n$. We only discuss our algorithm for computing $T_L(x', s_L^i)$ for $i = 1, \ldots, n$ since the algorithms for the other three cases are quite similar. Further, for each $1 \leqslant i \leqslant n$, the function $f_L^j(x', s_L^i)$ that gives the value $T_L(x', s_L^i)$ is also determined by the algorithm.

For any $1 \leqslant i \leqslant j \leqslant n$, we define $\alpha(i,j) = \sum_{k=i}^{j}(w_k^+ - w_k^-)$. After $O(n)$ time preprocessing, given any i and j with $1 \leqslant i \leqslant j \leqslant n$, we can obtain the value $\alpha(i,j)$ in constant time. We omit the preprocessing details and below we assume we have done the preprocessing. For convenience, we let $\alpha(i,j) = 0$ if $i > j$.

Let x' be any value with $x_1 \leqslant x' \leqslant x_n$. We first determine the index i such that $x_{i-1} < x' \leqslant x_i$. Thus, for any scenario s, only functions $f_L^t(x', s)$ with $1 \leqslant t \leqslant i-1$ are defined on $x = x'$, and any function $f_L^t(x, s)$ with $i \leqslant t \leqslant n$ does not define on $x = x'$. We compute the value $T_L(x, s_L^1)$, which can be done in $O(n)$ time, e.g., by computing $f_L^j(x', s_L^1)$ for each j with $1 \leqslant j \leqslant i-1$.

Let k be the index such that $T_L(x', s_L^1)$ is given by the function $f_L^k(x', s_L^1)$, e.g., $T_L(x', s_L^1) = f_L^k(x', s_L^1)$. Hence, $k \leqslant i-1$. The following lemma will be useful later.

Lemma 6. *Consider a function $f_L^t(x,s)$ and a scenario s_L^j. If $1 \leqslant t, j \leqslant i-1$, then $f_L^t\left(x', s_L^j\right) = f_L^t(x', s_L^1) + \alpha(2, m)$ with $m = \min\{t, j\}$. This implies $f_L^t(x', s_L^t) = f_L^t\left(x', s_L^j\right)$ if $t \leqslant j \leqslant i-1$.*

Proof. Consider any t and j with $1 \leqslant t, j \leqslant i-1$. First of all, since $x_{i-1} < x' \leqslant x_i, t \leqslant i-1$, and $j \leqslant i-1$, both functions $f_L^t(x, s_L^1)$ and $f_L^t\left(x, s_L^j\right)$ are defined on $x = x'$. Comparing with the scenario s_L^1, the weight of each vertex v_h for $2 \leqslant h \leqslant j$ increase by $w_h^+ - w_h^-$ in the scenario s_L^j, and the weights of all other vertices are the same as before. According to their definitions, we obtain that $f_L^t(x', s_L^1) = f_L^t\left(x', s_L^j\right) + \alpha(2, t)$ if $t \leqslant j$, and $f_L^t(x', s_L^1) = f_L^t\left(x', s_L^j\right) + \alpha(2, j)$ if $t \geqslant j$. The lemma thus follows. $\square$

With the value $T_L(x', s_L^1)$, the following lemma shows how to compute $T_L\left(x', s_L^j\right)$ for $2 \leqslant j \leqslant k$.

Lemma 7. *If $k \geqslant 2$, for any scenario s_L^j with $2 \leqslant j \leqslant k$, $T_L\left(x', s_L^j\right) = T_L(x', s_L^1) + \alpha(2, j)$.*

Proof. Assume $k \geqslant 2$. Consider any scenario s_L^j with $2 \leqslant j \leqslant k$. We first prove a *claim* that $f_L^k\left(x', s_L^j\right) \geqslant f_L^t\left(x', s_L^j\right)$ for any $1 \leqslant t \leqslant i-1$.

Due to $T_L(x', s_L^1) = f_L^k(x', s_L^1)$, it holds that $f_L^k(x', s_L^1) \geqslant f_L^t(x', s_L^1)$ for any $1 \leqslant t \leqslant i-1$. Consider any t with $1 \leqslant t \leqslant i-1$. By Lemma 6, we have $f_L^t\left(x', s_L^j\right) = f_L^t(x', s_L^1) + \alpha(2, m)$, where $m = \min\{j, t\}$. Since $j \leqslant k$, $f_L^k\left(x', s_L^j\right) = f_L^k(x', s_L^1) + \alpha(2, j)$ holds by Lemma 6. Clearly, $\alpha(2, j) \geqslant \alpha(2, m) \geqslant 0$ due to $m \leqslant j$. Therefore, we obtain that $f_L^k\left(x', s_L^j\right) \geqslant f_L^t\left(x', s_L^j\right)$.

The above claim implies that $T_L\left(x', s_L^j\right) = f_L^k\left(x', s_L^j\right)$. Since $f_L^k\left(x', s_L^j\right) = f_L^k(x', s_L^1) + \alpha(2, j)$ by Lemma 6 and $T_L(x', s_L^1) = f_L^k(x', s_L^1)$, the lemma follows. $\square$

Suppose the value $T_L(x', s_L^{i-1})$ has already been computed; the following lemma shows how to obtain $T_L\left(x', s_L^j\right)$ for $i \leqslant j \leqslant n$.

Lemma 8. *For any scenario s_L^j with $i \leqslant j \leqslant n$, $T_L\left(x', s_L^j\right) = T_L(x', s_L^{i-1})$.*

Proof. Recall that for any scenario s only the functions $f_L^t(x, s)$ with $1 \leqslant t \leqslant i-1$ are defined on $x = x'$. Consider any scenario s_L^j with $i \leqslant j \leqslant n$. Comparing with s_L^{i-1}, the weight of each vertex v_t in s_L^j increases by $w_t^+ - w_t^-$ for any $i \leqslant t \leqslant j$, and all other vertex weights do not change. Since the above vertex weight increase only affect the functions $f_L^t\left(x, s_L^j\right)$ for $t \geqslant i$ and none of these functions is defined on $x = x'$, the value $T_L(x', s)$ does not change for $s = s_L^{i-1}$ and $s = s_L^j$. A more formal proof is given below.

Let k' be the index such that the value $T_L(x', s_L^{i-1})$ is given by $f_L^{k'}(x', s_L^{i-1})$, i.e., $T_L(x', s_L^{i-1}) = f_L^{k'}(x', s_L^{i-1})$. Note that $k' \leqslant i-1$. Hence, $f_L^{k'}(x', s_L^{i-1}) \geqslant f_L^t(x', s_L^{i-1})$ for any $1 \leqslant t \leqslant i-1$. In the scenario s_L^j, the weights of the vertices v_t for $1 \leqslant t \leqslant i-1$ are the same as those in s_L^{i-1}. Therefore, $f_L^t(x', s_L^{i-1}) = f_L^t\left(x', s_L^j\right)$ for any $1 \leqslant t \leqslant i-1$. Thus, $f_L^{k'}\left(x', s_L^j\right) \geqslant f_L^t\left(x', s_L^j\right)$ for any $1 \leqslant t \leqslant i-1$. We obtain that $T_L\left(x', s_L^j\right) = f_L^{k'}\left(x', s_L^j\right)$. Due to $f_L^{k'}\left(x', s_L^j\right) = f_L^{k'}(x', s_L^{i-1})$ and $T_L(x', s_L^{i-1}) = f_L^{k'}(x', s_L^{i-1})$, we have $T_L\left(x', s_L^j\right) = T_L(x', s_L^{i-1})$. $\square$

Based on the preceding two lemmas, we can easily compute $T_L\left(x', s_L^j\right)$ for $j = 2, \ldots, k$ in $O(n)$ time, and compute $T_L\left(x', s_L^j\right)$ for $j = i, \ldots, n$ in $O(n)$ time provided that we know the value $T_L(x', s_L^{i-1})$.

It remains to compute $T_L(x', s_L^t)$ for $t = k+1, \ldots, i-1$, for which we present an $O(n)$ time algorithm below. Note that our algorithm itself is simple (see the pseudocode Algorithm 1), but it is not easy to discover the observations behind the scene. Our algorithm will compute a *solution index list* $K = \{k_1, k_2, \ldots, k_d\}$ with the following properties:

Property 1. *$k_1 = k$ and $k_1 \leqslant k_2 \leqslant \cdots \leqslant k_d \leqslant i-1$.*

Property 2. *For any j with $1 \leqslant j \leqslant d-1$, $f_L^{k_j}\left(x', s_L^{k_j}\right) < f_L^{k_{j+1}}\left(x', s_L^{k_{j+1}}\right)$ and $f_L^{k_j}(x', s_L^1) \geqslant f_L^{k_{j+1}}(x', s_L^1)$.*

Property 3. *For any j with $1 \leqslant j \leqslant d-1$, for any t with $k_j \leqslant t < k_{j+1}$, either $f_L^t(x', s_L^t) \leqslant f_L^{k_j}\left(x', s_L^{k_j}\right)$ or $f_L^t(x', s_L^1) < f_L^{k_{j+1}}(x', s_L^1)$. If $k_d \neq i-1$, then for any t with $k_d \leqslant t \leqslant i-1$, $f_L^t(x', s_L^t) \leqslant f_L^{k_d}\left(x', s_L^{k_d}\right)$.*

If we already have such a solution index list K, the lemma below provides a way to compute the values $T_L(x', s_L^t)$ for $k+1 \leqslant t \leqslant i-1$ in $O(n)$ time.

Lemma 9. *For any scenario s_L^t with $k+1 \leqslant t \leqslant i-1$, if $k_j < t \leqslant k_{j+1}$ for some $1 \leqslant j \leqslant d-1$, then $T_L(x', s_L^t) = \max\left\{f_L^{k_j}(x', s_L^t), f_L^{k_{j+1}}(x', s_L^t)\right\}$; if $k_d \neq i-1$ and $k_d < t$, then $T_L(x', s_L^t) = f_L^{k_d}(x', s_L^t)$.*

Proof. Consider any scenario s_L^t with $k+1 \leqslant t \leqslant i-1$. Recall that $T_L(x', s_L^t) = \max_{1 \leqslant m \leqslant i-1} f_L^m(x', s_L^t)$. To simplify the notation, we use $f^m(s^t)$ to represent $f_L^m(x', s_L^t)$. Hence, $T_L(x', s_L^t) = \max_{1 \leqslant m \leqslant i-1} f^m(s^t)$.

We assume $t \leqslant k_d$ since the case $t > k_d$ can be proved in a much simpler way by the same techniques. Let j be the integer such that $k_j < t \leqslant k_{j+1}$. To prove the lemma, it is sufficient to show that $\max\{f^{k_j}(s^t), f^{k_{j+1}}(s^t)\} \geqslant f^m(s^t)$ for any m with $1 \leqslant m \leqslant i-1$. To this

end, there are three cases depending on the value of $m : 1 \leqslant m < k_j, k_j \leqslant m < k_{j+1}$, and $k_{j+1} \leqslant m \leqslant i-1$. Below, in each case, we will show that either $f^m(s^t) \leqslant f^{k_j}(s^t)$ or $f^m(s^t) \leqslant f^{k_{j+1}}(s^t)$ holds.

First of all, due to $k_j < t$ and by Lemma 6, the following holds

$$f^{k_j}(s^t) = f^{k_j}(s^{k_j}). \tag{1}$$

1. If $1 \leqslant m < k_j$, we assume $k_h \leqslant m < k_{h+1}$ for some $h < j$. Note that $m < t$ holds in this case. See Fig. 5. By Lemma 6, we have $f^m(s^t) = f^m(s_L^m)$. By Property 3 of the solution index list K, we have either $f^m(s^m) \leqslant f^{k_h}(s^{k_h})$ or $f^m(s^1) < f^{k_{h+1}}(s^1)$.
   (a) If $f^m(s^m) \leqslant f^{k_h}(s^{k_h})$, then by Property 2 of K, since $h < j$, we can obtain $f^{k_h}(s^{k_h}) < f^{k_{h+1}}(s^{k_{h+1}}) < \cdots < f^{k_j}(s^{k_j})$. Thus, we have $f^m(s^t) = f^m(s^m) < f^{k_j}(s^{k_j})$. Since $f^{k_j}(s^t) = f^{k_j}(s^{k_j})$ by Eq. (1), we obtain $f^m(s^t) < f^{k_j}(s^t)$.
   (b) If $f^m(s^1) < f^{k_{h+1}}(s^1)$, then since $m < k_{h+1}$, we have $f^m(s^m) = f^m(s^1) + \alpha(2,m)$ and $f^{k_{h+1}}(s^m) = f^{k_{h+1}}(s^1) + \alpha(2,m)$, and thus $f^m(s^m) < f^{k_{h+1}}(s^m)$. Note that $f^{k_{h+1}}(s^m) \leqslant f^{k_{h+1}}(s^{k_{h+1}})$. Further, due to $k_{h+1} \leqslant k_j, f^{k_{h+1}}(s^{k_{h+1}}) < f^{k_j}(s^{k_j})$ holds by Property 2 of K. Recall that $f^m(s^t) = f^m(s_L^m)$. Hence, we obtain $f^m(s^t) = f^m(s_L^m) < f^{k_j}(s^{k_j}) = f^{k_j}(s^t)$ by Eq. (1).
2. If $k_j \leqslant m < k_{j+1}$, then by Property 3 of K, either $f^m(s^m) \leqslant f^{k_j}(s^{k_j})$ or $f^m(s^1) < f^{k_{j+1}}(s^1)$.
   (a) If $f^m(s^m) \leqslant f^{k_j}(s^{k_j})$, then since $f^m(s^t) \leqslant f^m(s^m)$ always holds by Lemma 6 regardless of whether $m \leqslant t$ or $m > t$, we have $f^m(s^t) \leqslant f^{k_j}(s^{k_j}) = f^{k_j}(s^t)$ by Eq. (1).
   (b) If $f^m(s^1) < f^{k_{j+1}}(s^1)$, then since $t \leqslant k_{j+1}$, we have $f^{k_{j+1}}(s^t) = f^{k_{j+1}}(s^1) + \alpha(2,t)$ by Lemma 6. Also, $f^m(s^t) = f^m(s^1) + \alpha(2, \min\{t,m\})$. Since $f^m(s^1) < f^{k_{j+1}}(s^1)$ and $\alpha(2,\min\{t,m\}) \leqslant \alpha(2,t)$, we have $f^m(s^t) \leqslant f^{k_{j+1}}(s^t)$.
3. If $k_{j+1} \leqslant m \leqslant i-1$, for simplicity of discussion, we assume $m < k_d$ and the case $m \geqslant k_d$ can be proved very similarly but in a much simpler way. Let $k_h \leqslant m < k_{h+1}$ for some $h > j$. Note that $t \leqslant k_{j+1} \leqslant k_h \leqslant m$. See Fig. 6.

   First of all, we claim that $f^m(s^t) \leqslant f^{k_h}(s^t)$. We prove the claim below.
   Indeed, by Lemma 6, we can obtain $f^m(s^m) = f^m(s^t) + \alpha(t+1,m)$ and $f^{k_h}(s^{k_h}) = f^{k_h}(s^t) + \alpha(t+1,k_h)$. According to Property 3 of K, either $f^m(s^m) \leqslant f^{k_h}(s^{k_h})$ or $f^m(s^1) < f^{k_{h+1}}(s^1)$.
   (a) If $f^m(s^m) \leqslant f^{k_h}(s^{k_h})$, then since $\alpha(t+1,m) \geqslant \alpha(t+1,k_h)$ (due to $k_h \leqslant m$), we obtain $f^m(s^t) \leqslant f^{k_h}(s^t)$.
   (b) If $f^m(s^1) < f^{k_{h+1}}(s^1)$, then by Property 2 of K, $f^{k_{h+1}}(s^1) \leqslant f^{k_h}(s^1)$. Thus, we obtain $f^m(s^1) \leqslant f^{k_h}(s^1)$. Due to $t \leqslant k_h \leqslant m$, we have $f^m(s^t) = f^m(s^1) + \alpha(2,t)$ and $f^{k_h}(s^t) = f^{k_h}(s^1) + \alpha(2,t)$. Hence, $f^m(s^t) \leqslant f^{k_h}(s^t)$ holds.

Therefore, the claim $f^m(s^t) \leqslant f^{k_h}(s^t)$ is proved.
   (a) If $j+1 = h$, the above proves $f^m(s^t) \leqslant f^{k_{j+1}}(s^t)$.
   (b) If $j+1 < h$, we claim that $f^{k_h}(s^t) \leqslant f^{k_{j+1}}(s^t)$. Indeed, since $t \leqslant k_{j+1} \leqslant k_h$ in this case, by Lemma 6, $f^{k_h}(s^t) = f^{k_h}(s^1) + \alpha(2,t)$ and $f^{k_{j+1}}(s^t) = f^{k_{j+1}}(s^1) + \alpha(2,t)$. By Property 2 of $K, f^{k_h}(s^1) \leqslant f^{k_{j+1}}(s^1)$. Therefore, $f^{k_h}(s^t) \leqslant f^{k_{j+1}}(s^t)$ and the claim is proved. Since $f^m(s^t) \leqslant f^{k_h}(s^t)$, we obtain $f^m(s^t) \leqslant f^{k_{j+1}}(s^t)$.

In any case above, we have shown that $f^m(s^t) \leqslant \max\{f^{k_j}(s^t), f^{k_{j+1}}(s^t)\}$ holds. The lemma thus follows. □

Suppose we have a solution index list K. After we compute the values $f_L^{k_j}(x', s_L^1)$ for $j = 1, \ldots, d$ in $O(n)$ time, by Lemma 9 we can compute the values $T_L(x', s_L^t)$ for all $k+1 \leqslant t \leqslant i-1$ in $O(n)$ time (with the help of Lemma 6).

It remains to compute the solution index list K, for which we present a simple linear time algorithm as follows. We assume

Fig. 5. $k_h \leqslant m < k_{h+1} \leqslant k_j < t \leqslant k_{j+1}$.

the values $f_L^t(x', s_L^1)$ for $t = 1, \ldots, i-1$ have been computed in $O(n)$ time.

Our algorithm will consider the indices incrementally from $t = k+1$ to $t = i-1$. A stack A is maintained during the algorithm to store a sequence of indices. Initially A contains only one index k, and after the algorithm finishes the index list in A from bottom to top is exactly K. Algorithm 1 summarizes the pseudocode. For each t with $k+1 \leqslant t \leqslant i-1$, the algorithm proceeds as follows. Let k_j be the index on the top of the current stack A. We first compare the two values $f_L^t(x', s_L^1)$ and $f_L^{k_j}(x', s_L^1)$. Note that both values have been computed. If $f_L^t(x', s_L^1) > f_L^{k_j}(x', s_L^1)$, then we pop k_j out of A and consider the next top index on A (this is consistent with Property 3 of K and we ignore the detailed discussion on this). We claim that the stack A will never be empty because the index k is at the bottom of A. Indeed, recall that $T_L(x', s_L^1) = f_L^k(x', s_L^1)$ by the definition of k. Hence, $f_L^k(x', s_L^1) \geqslant f_L^m(x', s_L^1)$ for any $1 \leqslant m \leqslant i-1$, and in particular, $f_L^t(x', s_L^1) \leqslant f_L^k(x', s_L^1)$. Therefore, k will never be popped out of A. If $f_L^t(x', s_L^1) \leqslant f_L^{k_j}(x', s_L^1)$, we further compare the two values $f_L^t(x', s_L^t)$ and $f_L^{k_j}\left(x', s_L^{k_j}\right)$. By Lemma 6, $f_L^t(x', s_L^t) = f_L^t(x', s_L^1) + \alpha(2,t)$ and $f_L^{k_j}\left(x', s_L^{k_j}\right) = f_L^{k_j}(x', s_L^1) + \alpha(2,k_j)$. Hence, both $f_L^t(x', s_L^t)$ and $f_L^{k_j}\left(x', s_L^{k_j}\right)$ can be computed in constant time. If $f_L^t(x', s_L^t) > f_L^{k_j}\left(x', s_L^{k_j}\right)$, then we push t on the top of A and set $k_{j+1} = t$ (this is consistent with Property 2 of K); otherwise, we ignore t (this is consistent with Property 3 of K) and proceed on $t+1$. After $t = i-1$ is considered, we terminate the algorithm and the index list in the stack A is our solution index list K. The running time of the algorithm is $O(n)$ because once an index is popped out of A it will never be considered again.

Algorithm 1. Computing the solution index list $K = \{k_1, \ldots, k_d\}$

Input: The index k, the value x', the scenario s_L^1, the weight intervals $[w_i^-, w_i^+]$
Output: The solution index list $K = \{k_1, \ldots, k_d\}$

1 do preprocessing for answering $\alpha(\cdot,\cdot)$ queries;
2 **for** $t \leftarrow 1$ **to** $i-1$ **do**
3 determine the function $f_L^t(x, s_L^1)$ and compute the value $f_L^t(x', s_L^1)$;
4 **end**
5 initialize a stack A and push k into A;
6 **for** $t \leftarrow k+1$ **to** $i-1$ **do**
7 $m \leftarrow$ the top index in A;
8 **while** $f_L^t(x', s_L^1) > f_L^m(x', s_L^1)$ **do**
9 pop m out of A;
10 $m \leftarrow$ the top index in A;
11 **end**
12 $f_L^t(x', s_L^t) \leftarrow f_L^t(x', s_L^1) + \alpha(2,t)$;
13 $f_L^m(x', s_L^m) \leftarrow f_L^m(x', s_L^1) + \alpha(2,m)$;
14 **if** $f_L^t(x', s_L^t) > f_L^m(x', s_L^m)$ **then**
15 push t on the top of A;
16 **end**
17 **end**
18 return the index list in A from bottom to top as K;

We conclude this section with the following theorem.

Fig. 6. $k_j < t \leqslant k_{j+1} \leqslant k_h \leqslant m < k_{h+1}$.

Theorem 2. *The optimal position x^* for the minmax regret problem and the optimal maximum regret $R_{max}(x^*)$ can be computed in $O(n \log n)$ time and $O(n)$ space.*

5. Concluding remarks

In this paper, we presented an $O(n \log n)$ time and $O(n)$ space algorithm for the minmax regret 1-facility location problem on uncertain path networks. Very recently some other problem variations have been introduced and algorithms for them have been proposed. Li, Xu, and Ni (2014) studied the problem for finding two facilities and gave an $O(n^3 \log n)$ time algorithm. Ni, Xu, and Dong (2014) developed an $O(n^{1+k} \log^{1+\log k} n)$ time algorithm for finding k facilities for a general value of k, and Arumugam, Augustine, Golin, and Srikanthan (2014) gave two algorithms for the same problem with time complexities $O(kn^2 \log^k n)$ and $O(kn^3 \log n)$, respectively. Higashikawa, Golin, and Katoh (2014) investigated the problem for finding a 1-facility on a tree network and proposed an $O(n \log^2 n)$ time algorithm. It would be interesting to see whether the techniques presented in this paper can be used for solving these problem variations.

Acknowledgment

The author wishes to thank the anonymous referees for their helpful comments leading to the improvement of the paper. This work was supported in part by NSF under Grant CCF-1317143.

References

Arumugam, G. P., Augustine, J., Golin, M., & Srikanthan, P. (2014). A polynomial time algorithm for minimax-regret evacuation on a dynamic path 1404.5448v1, April.

Averbakh, I., & Bereg, S. (2005). Facility location problems with uncertainty on the plane. *Discrete Optimization, 2*, 3–34.

Averbakh, I., & Berman, O. (1997). Minmax regret *p*-center location on a network with demand uncertainty. *Location Science, 5*, 247–254.

Averbakh, I., & Berman, O. (2000a). Algorithms for the robust 1-center problem on a tree. *European Journal of Operational Research, 123*, 292–302.

Averbakh, I., & Berman, O. (2000b). Minmax regret median location on a network under uncertainty. *INFORMS Journal on Computing, 12*, 104–110.

Averbakh, I., & Berman, O. (2003). An improved algorithm for the minmax regret median problem on a tree. *Networks, 2*, 97–103.

Bhattacharya, B., & Kameda, T. (2012). A linear time algorithm for computing minmax regret 1-median on a tree. In *Proceedings of the 18th annual international conference on computing and combinatorics* (pp. 1–12).

Bhattacharya, B., Kameda, T., & Song, Z. (2012). Computing minmax regret 1-median on a tree network with positive/negative vertex weights. In *Proceedings of the 23rd international symposium on algorithms and computation* (pp. 588–597).

Bhattacharya, B., Kameda, T., & Song, Z. (2012). Minmax regret 1-center on a path/cycle/tree. In *Proceedings of the sixth international conference on advanced engineering computing and applications in sciences* (pp. 108–113).

Chen, B., & Lin, C.-S. (1998). Minmax-regret robust 1-median location on a tree. *Networks, 31*, 93–103.

Cheng, S.-W., Higashikawa, Y., Katoh, N., Ni, G., Su, B., & Xu, Y. (2013). Minimax regret 1-sink location problems in dynamic path networks. In *Proceedings of the 10th annual conference on theory and applications of models of computation* (pp. 121–132).

Conde, E. (2007). Minmax regret location allocation problem on a network under uncertainty. *European Journal of Operational Research, 179*, 1025–1039.

Conde, E. (2008). A note on the minmax regret centdian location on trees. *Operations Research Letters, 36*, 271–275.

Driscoll, J., Sarnak, N., Sleator, D., & Tarjan, R. E. (1989). Making data structures persistent. *Journal of Computer and System Sciences, 38*(1), 86–124.

Higashikawa, Y., Auguestine, J., Cheng, S.-W., Golin, M. J., Katoh, N., Ni, G., et al. (2014). Minimax regret 1-sink location problems in dynamic path networks. *Theoretical Computer Science*. Available online.

Higashikawa, Y., Golin, M. J., & Katoh, N. (2014). Minimax regret sink location problem in dynamic tree networks with uniform capacity. In *Proceedings of the 8th international workshop on algorithms and computation (WALCOM)* (pp. 125–137).

Kamiyama, N., Katoh, N., & Takizawa, A. (2006). An efficient algorithm for evacuation problem in dynamic network flows with uniform arc capacity. *Transactions on Information and Systems, E89-D*(8), 2372–2379.

Kouvelis, P., & Yu, G. (Eds.). (1997). *Robust discrete optimization and its applications*. Dordrecht: Kluwer Academic Publishers.

Li, H., Xu, Y., & Ni, G. (2014). Minimax regret vertex 2-sink location problem in dynamic path networks. *Journal of Combinatorial Optimization*. Available online.

Megiddo, N. (1983). Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing, 12*(4), 759–776.

Megiddo, N. (1984). Linear programming in linear time when the dimension is fixed. *Journal of the ACM, 31*(1), 114–127.

Ni, G., Xu, Y., & Dong, Y. (2014). Minimax regret k-sink location problem in dynamic path networks. In *Proceedings of the 10th international conference on algorithmic aspects of information and management (AAIM)* (pp. 23–31).

Puerto, J., Rodríguez-Chía, A. M., & Tamir, A. (2009). Minimax regret single-facility ordered median location problems on networks. *INFORMS Journal on Computing, 21*, 77–87.

Wang, H. (2013). Minmax regret 1-facility location on uncertain path networks. In *Proceedings of the 24th international symposium on algorithms and computation (ISAAC)* (pp. 733–743).

Yu, H.-I., Lin, T.-C., & Wang, B.-F. (2008). Improved algorithms for the minmax-regret 1-center and 1-median problems. *ACM Transactions on Algorithms, 4*(3). Article No. 36.