Theoretical Computer Science 588 (2015) 24–36

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Minimax regret 1-sink location problem in dynamic path networks ☆

Yuya Higashikawa [a,*], John Augustine [b], Siu-Wing Cheng [c,1], Mordecai J. Golin [c], Naoki Katoh [a,2], Guanqun Ni [d], Bing Su [e,g], Yinfeng Xu [d,f,g,3]

[a] Department of Architecture and Architectural Engineering, Kyoto University, Japan
[b] Department of Computer Science and Engineering, Indian Institute of Technology Madras, India
[c] Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong
[d] Business School, Sichuan University, Chengdu 610065, China
[e] School of Economics and Management, Xi'an Technological University, Xi'an 710032, China
[f] School of Management, Xi'an Jiaotong University, Xi'an 710049, China
[g] State Key Lab for Manufacturing Systems Engineering, Xi'an 710049, China

ARTICLE INFO

Article history:
Received 30 September 2013
Received in revised form 15 January 2014
Accepted 7 February 2014
Available online 14 February 2014

Keywords:
Minimax regret
Sink location
Dynamic flow
Evacuation planning

ABSTRACT

This paper considers the minimax regret 1-sink location problem in dynamic path networks. In our model, a dynamic path network consists of an undirected path with positive edge lengths and uniform edge capacity, and each vertex supply which is nonnegative value is unknown but only the interval of supply is known. A particular assignment of supply to each vertex is called a scenario. Under any scenario, the cost of a sink location is defined as the minimum time to complete the evacuation for all supplies (evacuees), and the regret of a sink location x is defined as the cost of x minus the cost of the optimal sink location. Then, the problem is to find a point as a sink such that the maximum regret for all possible scenarios is minimized. We propose an $O(n \log n)$ time algorithm for the minimax regret 1-sink location problem in dynamic path networks with uniform capacity, where n is the number of vertices in the network.

1. Introduction

The Tohoku–Pacific Ocean Earthquake happened in Japan on March 11, 2011, and many people failed to evacuate and lost their lives due to severe attack by tsunamis. From the viewpoint of disaster prevention from city planning and evacuation planning, it has now become extremely important to establish effective evacuation planning systems against large scale disasters. In particular, arrangements of tsunami evacuation buildings in large Japanese cities near the coast have become an urgent issue. To determine appropriate tsunami evacuation buildings, we need to consider where evacuation buildings are assigned and how to partition a large area into small regions so that one evacuation building is designated in each region. This produces several theoretical issues to be considered. Among them, this paper focuses on the location problem

☆ A preliminary version of this paper appeared in the proceedings of TAMC 2013 [5].

* Corresponding author.
E-mail address: as.higashikawa@archi.kyoto-u.ac.jp (Y. Higashikawa).

[1] Supported by the Research Grants Council, Hong Kong, China (611812).
[2] Supported by JSPS Grant-in-Aid for Scientific Research (A) (25240004).
[3] Supported by the National Natural Science Foundation of China under Grants 71071123, 61221063 and Program for Changjiang Scholars and Innovative Research Team in University under Grant IRT1173.

http://dx.doi.org/10.1016/j.tcs.2014.02.010
0304-3975/

of the evacuation building assuming that we fix the region such that all evacuees in the region are planned to evacuate to this building. In this paper, we consider the simplest case for which the region consists of a single road.

An evaluation criterion of the building location is the time required to complete the evacuation. In order to represent the evacuation, we consider the *dynamic* setting in graph networks, which was first introduced by Ford et al. [8]. Under the dynamic setting, each edge of a given graph has capacity which limits the rate of the flow into the edge per unit time. We call such networks under the dynamic setting *dynamic networks*. If a sink location is given in a dynamic network with the initial supplies on vertices, the *cost* of the sink location is defined as the minimum time to complete the evacuation to the sink for all supplies. Then, *the minimax cost 1-sink location problem in dynamic networks* is defined as the problem to find a sink location in a given network which minimizes the cost. In this setting, initial supplies and a sink in a network represent evacuees and an evacuation center in a city, respectively. Mamada et al. [11] studied the minimax cost 1-sink location problem in dynamic tree networks and proposed an $O(n \log^2 n)$ time algorithm.

However, the initial supply at a vertex (the number of evacuees in an area) may vary depending on the time (e.g., in an office area in a big city there are many people during the daytime on weekdays while there are much less people on weekends or during the night time). So, in order to take into account the uncertainty of vertex supplies, we consider a *maximum regret* for a particular sink location as another evaluation criterion assuming that for each vertex, we only know the interval of vertex supply. In this paper, we formulate the problem as *the minimax regret 1-sink location problem in dynamic networks*. A particular realization (assignment of supply to each vertex) is called a *scenario*. The problem can be understood as a 2-person Stackelberg game as follows. The first player picks a sink location x and the second player chooses a scenario s that maximizes the *regret* which is defined as the cost of x (i.e., the minimum time to complete the evacuation to x) minus the cost of the optimal sink location under the scenario s. The objective of the first player is to choose x that minimizes the maximum regret. Recently, several researchers have studied the minimax regret facility location problems and proposed efficient algorithms [1–4,6,7,10,12,13]. In this paper, we propose an $O(n \log n)$ time algorithm for the minimax regret 1-sink location problem in dynamic path networks with uniform capacity.

2. Preliminaries

2.1. Definition

In the following, for two integers i and j, let $[i, j] = \{k \in \mathbb{Z} \mid i \leqslant k \leqslant j\}$. Let $P = (V, E)$ be a path where $V = \{v_0, v_1, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_n\}$ such that v_{i-1} and v_i are endpoints of e_i for $i \in [1, n]$. Let $\mathcal{N} = (P, l, W, c, \tau)$ be a dynamic flow network with the underlying undirected graph being a path P, where l is a function that associates each edge e_j with positive integral length $l(e_j)$, W is also a function that associates each vertex $v_i \in V$ with an interval of integral supply (corresponding to the number of the evacuees) $W(v_i) = [w^-(v_i), w^+(v_i)]$ with $0 < w^-(v_i) \leqslant w^+(v_i)$, c is an integral constant representing the capacity of each edge: the least upper bound for the number of the evacuees entering an edge per unit time, and τ is also an integral constant representing the time required for traversing the unit distance of each evacuee. We call such networks with path structures *dynamic path networks*. Let $\mathcal{S}$ denote the Cartesian product of all $W(v_i)$ for $i \in [0, n]$ (i.e., a set of scenarios):

$$\mathcal{S} = \prod_{i \in [0,n]} \left[w^-(v_i), w^+(v_i)\right]. \tag{1}$$

When a scenario $s \in \mathcal{S}$ is given, we use the notation $w(v_i, s)$ to denote the supply of each vertex $v_i \in V$ under the scenario s.

In the following, we abuse the notation P to denote the set of all points $p \in P$. Also, for a vertex $v_i \in V$ with $i \in [0, n]$, we abuse the notation v_i to denote the distance between v_0 and v_i in P, and for a point $p \in P$, we abuse the notation p to denote the distance between v_0 and p in P. Then, we can regard P as embedded on a real line such that $v_0 = 0$.

Suppose that a sink is located at a point $x \in P$. Let $\tilde{\Theta}_L(x, s)$ (resp. $\tilde{\Theta}_R(x, s)$) denote the minimum time required for all evacuees on the part of P consisting of all points $p \in P$ such that $p < x$ (resp. $x < p$) to complete the evacuation to x under a scenario $s \in \mathcal{S}$. Note that we assume that the capacity of the entrance of an evacuation building is infinite, and thus, if we place a sink in a vertex v_i, all evacuees of v_i can finish their evacuation in no time. Then, by [9], $\tilde{\Theta}_L(x, s)$ and $\tilde{\Theta}_R(x, s)$ can be expressed as follows:

$$\Theta_L(x, s) = \max_{i \in [0, n-1]} \left\{ (x - v_i)\tau + \left\lceil \frac{\sum_{j \in [0,i]} w(v_j, s)}{c} \right\rceil - 1 \;\middle|\; v_i < x \right\}, \quad \text{and}$$

$$\Theta_R(x, s) = \max_{i \in [1, n]} \left\{ (v_i - x)\tau + \left\lceil \frac{\sum_{j \in [i,n]} w(v_j, s)}{c} \right\rceil - 1 \;\middle|\; v_i > x \right\}.$$

In the following discussion, we assume $c = 1$ and for the ease of exposition, omit the constant term (i.e., -1) from the above equations. Thus, we redefine $\Theta_L(x, s)$ and $\Theta_R(x, s)$ as

$$\Theta_L(x, s) = \max_{i \in [0, n-1]} \left\{ (x - v_i)\tau + \sum_{j \in [0,i]} w(v_j, s) \;\middle|\; v_i < x \right\}, \quad \text{and} \tag{2}$$

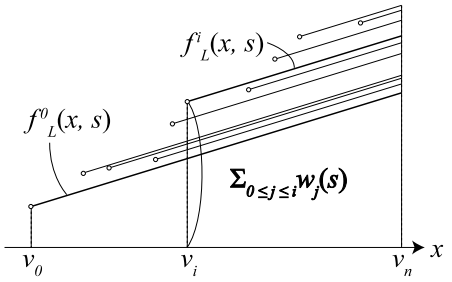

Fig. 1. Functions $f^i_L(x, s)$ for $i \in [0, n-1]$.

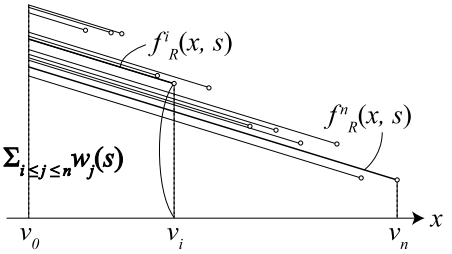

Fig. 2. Functions $f^i_R(x, s)$ for $i \in [1, n]$.

$$\Theta_R(x, s) = \max_{i \in [1,n]} \left\{ (v_i - x)\tau + \sum_{j \in [i,n]} w(v_j, s) \;\middle|\; v_i > x \right\}. \tag{3}$$

Additionally, we regard $\Theta_L(v_0, s)$ and $\Theta_R(v_n, s)$ as 0 in the subsequent discussion. Now, under $s \in \mathcal{S}$, the minimum time required for the evacuation to $x \in P$ of all evacuees is defined by

$$\Theta(x, s) = \max\{\Theta_L(x, s), \Theta_R(x, s)\}. \tag{4}$$

Let $f^i_L(x, s)$ and $f^i_R(x, s)$ denote functions defined as follows: for $i \in [0, n-1]$,

$$f^i_L(x, s) = (x - v_i)\tau + \sum_{j \in [0,i]} w(v_j, s) \quad (x > v_i), \tag{5}$$

and for $i \in [1, n]$,

$$f^i_R(x, s) = (v_i - x)\tau + \sum_{j \in [i,n]} w(v_j, s) \quad (x < v_i). \tag{6}$$

Then, $\Theta_L(x, s)$ and $\Theta_R(x, s)$ are expressed as follows:

$$\Theta_L(x, s) = \max_{i \in [0,n-1]} \left\{ f^i_L(x, s) \;\middle|\; v_i < x \right\}, \quad \text{and} \tag{7}$$

$$\Theta_R(x, s) = \max_{i \in [1,n]} \left\{ f^i_R(x, s) \;\middle|\; v_i > x \right\}. \tag{8}$$

The function $f^i_L(x, s)$ is drawn as a left-open segment with a positive slope τ starting from $(v_i, \sum_{j \in [0,i]} w(v_j, s))$ and ending at $(v_n, (v_n - v_i)\tau + \sum_{j \in [0,i]} w(v_j, s))$ (see Fig. 1) while $f^i_R(x, s)$ is drawn as a right-open segment with a negative slope $-\tau$ starting from $(v_0, (v_i - v_0)\tau + \sum_{j \in [i,n]} w(v_j, s))$ and ending at $(v_i, \sum_{j \in [i,n]} w(v_j, s))$ (see Fig. 2). Thus $\Theta_L(x, s)$ is the upper envelope of these n segments, and so $\Theta_L(x, s)$ is a strictly monotone increasing function of x. Symmetrically, $\Theta_R(x, s)$ is a strictly monotone decreasing function of x. Therefore, $\Theta(x, s)$ is a unimodal function, so there is a unique point in P which minimizes $\Theta(x, s)$ (see Fig. 3). In the following, let $x_{\mathrm{opt}}(s)$ denote such a point in P:

$$x_{\mathrm{opt}}(s) = \operatorname{argmin}\{\Theta(x, s) \mid v_0 \leqslant x \leqslant v_n\}. \tag{9}$$

We have the following proposition.

Proposition 1. *Under a scenario $s \in \mathcal{S}$,*

(i) *$x_{\mathrm{opt}}(s)$ is unique,*

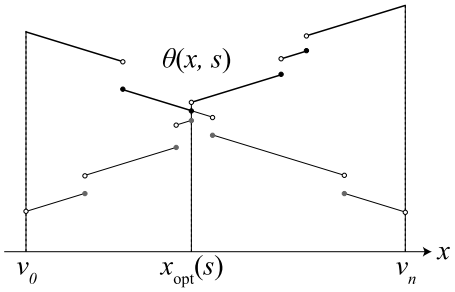

Fig. 3. A function $\Theta(x, s)$.

(ii) *for a sink location* $x < x_{\mathrm{opt}}(s)$, $\Theta_L(x, s) < \Theta_R(x, s)$ *holds, and*
(iii) *for a sink location* $x > x_{\mathrm{opt}}(s)$, $\Theta_L(x, s) > \Theta_R(x, s)$ *holds.*

Note that Proposition 1(ii), (iii) implies that $\Theta(x, s) = \Theta_R(x, s)$ holds for $x < x_{\mathrm{opt}}(s)$ and $\Theta(x, s) = \Theta_L(x, s)$ holds for $x > x_{\mathrm{opt}}(s)$.

In the following, we use the notation $\Theta_{\mathrm{opt}}(s)$ for a scenario $s \in \mathcal{S}$ to denote $\Theta(x_{\mathrm{opt}}(s), s)$:

$$\Theta_{\mathrm{opt}}(s) = \min\{\Theta(x, s) \mid v_0 \leqslant x \leqslant v_n\}. \tag{10}$$

We define the *regret* for x under s as

$$R(x, s) = \Theta(x, s) - \Theta_{\mathrm{opt}}(s). \tag{11}$$

Moreover, we also define the *maximum regret* for x as

$$R_{\max}(x) = \max\{R(x, s) \mid s \in \mathcal{S}\}. \tag{12}$$

For a sink location x, if $\hat{s} = \mathrm{argmax}\{R(x, s) \mid s \in \mathcal{S}\}$, we call $\hat{s}$ the *worst case scenario* for x. The goal is to find a sink location x^* which minimizes $R_{\max}(x)$, called the *minimax regret sink location*. Therefore, the minimax regret 1-sink location problem in a path P is defined as follows:

$$\text{minimize } \{R_{\max}(x) \mid x \in P\}. \tag{13}$$

2.2. Properties

In this section, we show some key properties of our problem. First, let us recall the following claim.

Claim 1. *For a scenario* $s \in \mathcal{S}$, *a function* $\Theta(x, s)$ *is unimodal in* x.

In contraposition to Proposition 1(ii), (iii), we also have the following lemma.

Lemma 1. *For a scenario* $s \in \mathcal{S}$ *and a sink location* $x \in P$,

(i) *if* $\Theta_L(x, s) \geqslant \Theta_R(x, s)$ *holds,* $x_{\mathrm{opt}}(s) \leqslant x$ *holds, and*
(ii) *if* $\Theta_L(x, s) \leqslant \Theta_R(x, s)$ *holds,* $x_{\mathrm{opt}}(s) \geqslant x$ *holds.*

For a given scenario $s \in \mathcal{S}$, by the definition (11) and Claim 1, a function $R(x, s)$ is also unimodal in x. Thus, a function $R_{\max}(x)$ is unimodal in x since it is the upper envelope of unimodal functions by (12).

Claim 2. *A function* $R_{\max}(x)$ *is unimodal in* x.

In the following, let x^* denote the minimax regret sink location in P. Then, we obtain the following lemma.

Lemma 2. *For a sink location* $x \in P$, *let* $\hat{s} = \mathrm{argmax}\{R(x, s) \mid s \in \mathcal{S}\}$. *Then,*

(i) *if* $\Theta_L(x, \hat{s}) \geqslant \Theta_R(x, \hat{s})$ *holds,* $x^* \leqslant x$ *holds, and*
(ii) *if* $\Theta_L(x, \hat{s}) \leqslant \Theta_R(x, \hat{s})$ *holds,* $x^* \geqslant x$ *holds.*

Proof. We only prove (i) by contradiction: suppose that $x^* > x$ and $\Theta_L(x, \hat{s}) \geqslant \Theta_R(x, \hat{s})$ hold. By Lemma 1(i), $x_{\mathrm{opt}}(\hat{s}) \leqslant x$ holds. Then, $\Theta(x^*, \hat{s}) > \Theta(x, \hat{s})$ holds by Claim 1, thus $R(x^*, \hat{s}) > R(x, \hat{s})$ also holds by (11). We have $R_{\max}(x^*) \geqslant R(x^*, \hat{s})$ by the maximality of $R_{\max}(x^*)$, and $R(x, \hat{s}) = R_{\max}(x)$ by the definition of $\hat{s}$. Thus, $R_{\max}(x^*) > R_{\max}(x)$ holds, which contradicts the optimality of x^*. □

For a scenario $s \in \mathcal{S}$ and an integer $i \in [0, n]$, let s^i_+ denote a scenario such that

$$w(v_i, s^i_+) = w^+(v_i) \quad \text{and} \quad w(v_j, s^i_+) = w(v_j, s) \quad \text{for } j \neq i, \tag{14}$$

and s^i_- denote a scenario such that

$$w(v_i, s^i_-) = w^-(v_i) \quad \text{and} \quad w(v_j, s^i_-) = w(v_j, s) \quad \text{for } j \neq i. \tag{15}$$

By (5), $f^i_L(x, s)$ is defined on $x > v_i$ with $i \in [0, n-1]$. Thus, for a sink location x such that $v_i < x \leqslant v_n$, $f^i_L(x, s) \leqslant f^i_L(x, s^i_+)$ and $f^i_L(x, s^i_-) \leqslant f^i_L(x, s)$ always hold for any i. Moreover, by these facts and (7), we also have $\Theta_L(x, s) \leqslant \Theta_L(x, s^i_+)$ and $\Theta_L(x, s^i_-) \leqslant \Theta_L(x, s)$. We have the following claim.

Claim 3. *For a scenario $s \in \mathcal{S}$, a sink location $x \in P$ and an integer $i \in [0, n]$ such that $v_0 \leqslant v_i \leqslant x$ (resp. $x \leqslant v_i \leqslant v_n$),*

(i) $\Theta_L(x, s) \leqslant \Theta_L(x, s^i_+)$ (resp. $\Theta_R(x, s) \leqslant \Theta_R(x, s^i_+)$), *and*
(ii) $\Theta_L(x, s^i_-) \leqslant \Theta_L(x, s)$ (resp. $\Theta_R(x, s^i_-) \leqslant \Theta_R(x, s)$) *holds.*

A scenario $s \in \mathcal{S}$ is said to be *left-dominant* (resp. *right-dominant*) if for a given $i \in [0, n]$, $w(v_j, s) = w^+(v_j)$ for $j \in [0, i]$ and $w(v_j, s) = w^-(v_j)$ for $j \in [i+1, n]$ hold (resp. $w(v_j, s) = w^-(v_j)$ for $j \in [0, i]$ and $w(v_j, s) = w^+(v_j)$ for $j \in [i+1, n]$ hold). Let $\mathcal{S}_L$ (resp. $\mathcal{S}_R$) denote the set of all left-dominant (resp. right-dominant) scenarios. $\mathcal{S}_L$ consists of the following $n+1$ scenarios:

$$\begin{aligned} s^i_L &= \big(w^+(v_0), \ldots, w^+(v_i), w^-(v_{i+1}), \ldots, w^-(v_n)\big) \quad \text{for } i \in [0, n-1], \quad \text{and} \\ s^n_L &= \big(w^+(v_0), w^+(v_1), \ldots, w^+(v_n)\big), \end{aligned} \tag{16}$$

and $\mathcal{S}_R$ consists of the following $n+1$ scenarios:

$$\begin{aligned} s^i_R &= \big(w^-(v_0), \ldots, w^-(v_i), w^+(v_{i+1}), \ldots, w^+(v_n)\big) \quad \text{for } i \in [0, n-1], \quad \text{and} \\ s^n_R &= \big(w^-(v_0), w^-(v_1), \ldots, w^-(v_n)\big). \end{aligned} \tag{17}$$

The following is a key lemma.

Lemma 3. *For a sink location $x \in P$, there exists a worst case scenario which belongs to $\mathcal{S}_L \cup \mathcal{S}_R$.*

Proof. Let $\hat{s} = \mathrm{argmax}\{R(x, s) \mid s \in \mathcal{S}\}$. We prove the lemma only for a sink location x such that $\Theta_L(x, \hat{s}) \geqslant \Theta_R(x, \hat{s})$. Suppose that $v_{k-1} < x \leqslant v_k$ with an integer $k \in [1, n]$ and $l = \mathrm{argmax}\{f^i_L(x, \hat{s}) \mid i \in [0, k-1]\}$, i.e.,

$$\Theta(x, \hat{s}) = f^l_L(x, \hat{s}) = (x - v_l)\tau + \sum_{j \in [0, l]} w(v_j, \hat{s}). \tag{18}$$

Then, we actually prove that $R(x, s^l_L) \geqslant R(x, \hat{s})$ holds. If $\hat{s}$ is not equal to s^l_L, we have two cases:

[Case 1] there exists an integer $i \in [0, l]$ such that $w(v_i, \hat{s}) < w^+(v_i)$, and
[Case 2] there exists an integer $i \in [l+1, n]$ such that $w(v_i, \hat{s}) > w^-(v_i)$.

If we can show that $R(x, \hat{s}^i_+) \geqslant R(x, \hat{s})$ holds for Case 1 and $R(x, \hat{s}^i_-) \geqslant R(x, \hat{s})$ holds for Case 2, we will eventually obtain $R(x, s^l_L) \geqslant R(x, \hat{s})$ by repeatedly applying the same discussion as long as there exists such a vertex v_i.

[Case 1]: Let $\Delta = w^+(v_i) - w(v_i, \hat{s})$. We first notice

$$\Theta(x, \hat{s}^i_+) = \Theta_L(x, \hat{s}^i_+) \quad \text{and} \tag{19}$$

$$\Theta_L(x, \hat{s}^i_+) = (x - v_l)\tau + \sum_{j \in [0, l]} w(v_j, \hat{s}^i_+) = \Theta(x, \hat{s}) + \Delta \tag{20}$$

by (2) and (18). Thus, we have

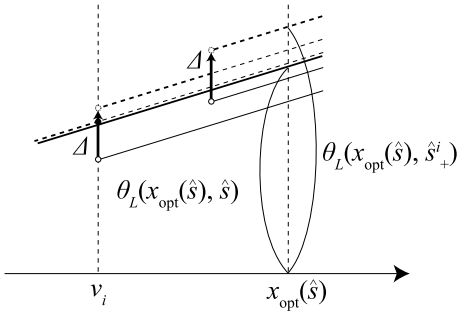

Fig. 4. Subcase of Case 1: $x_{\mathrm{opt}}(\hat{s}) > v_i$.

$$\Theta\left(x, \hat{s}^i_+\right) = \Theta(x, \hat{s}) + \Delta. \tag{21}$$

By the optimality of $x_{\mathrm{opt}}(\hat{s}^i_+)$ under $\hat{s}^i_+$ (see (9)), $\Theta_{\mathrm{opt}}(\hat{s}^i_+) \leqslant \Theta(x_{\mathrm{opt}}(\hat{s}), \hat{s}^i_+)$ holds. Here, we show

$$\Theta\left(x_{\mathrm{opt}}(\hat{s}), \hat{s}^i_+\right) \leqslant \Theta\left(x_{\mathrm{opt}}(\hat{s}), \hat{s}\right) + \Delta = \Theta_{\mathrm{opt}}(\hat{s}) + \Delta \tag{22}$$

for the subcase of $x_{\mathrm{opt}}(\hat{s}) > v_i$ (the other case can be treated similarly). In this case, $\Theta_L(x_{\mathrm{opt}}(\hat{s}), \hat{s}^i_+) \leqslant \Theta_L(x_{\mathrm{opt}}(\hat{s}), \hat{s}) + \Delta$ (see Fig. 4) and $\Theta_R(x_{\mathrm{opt}}(\hat{s}), \hat{s}^i_+) = \Theta_R(x_{\mathrm{opt}}(\hat{s}), \hat{s})$ hold by the definitions (2) and (3), which implies that (22) holds.

Thus, we have

$$\Theta_{\mathrm{opt}}\left(\hat{s}^i_+\right) \leqslant \Theta_{\mathrm{opt}}(\hat{s}) + \Delta. \tag{23}$$

By (11), (21) and (23), we obtain $R(x, \hat{s}^i_+) \geqslant R(x, \hat{s})$.

[Case 2]: In this case, $\Theta(x, \hat{s}^i_-) = \Theta_L(x, \hat{s}^i_-)$ and $\Theta_L(x, \hat{s}^i_-) = \Theta_L(x, \hat{s})$ by (2) and (18). Thus, we have

$$\Theta\left(x, \hat{s}^i_-\right) = \Theta(x, \hat{s}). \tag{24}$$

By the optimality of $x_{\mathrm{opt}}(\hat{s}^i_-)$ under $\hat{s}^i_-$, $\Theta_{\mathrm{opt}}(\hat{s}^i_-) \leqslant \Theta(x_{\mathrm{opt}}(\hat{s}), \hat{s}^i_-)$ holds. By Claim 3(ii) and (10), $\Theta(x_{\mathrm{opt}}(\hat{s}), \hat{s}^i_-) \leqslant \Theta(x_{\mathrm{opt}}(\hat{s}), \hat{s}) = \Theta_{\mathrm{opt}}(\hat{s})$ holds. Thus, we have

$$\Theta_{\mathrm{opt}}\left(\hat{s}^i_-\right) \leqslant \Theta_{\mathrm{opt}}(\hat{s}). \tag{25}$$

By (11), (24) and (25), we obtain $R(x, \hat{s}^i_-) \geqslant R(x, \hat{s})$. □

We also have the following lemma.

Lemma 4. *For a scenario $s \in \mathcal{S}$ and an integer $i \in [0, n]$ such that $v_0 \leqslant v_i \leqslant x_{\mathrm{opt}}(s)$ (resp. $x_{\mathrm{opt}}(s) \leqslant v_i \leqslant v_n$), $v_i \leqslant x_{\mathrm{opt}}(s^i_+) \leqslant x_{\mathrm{opt}}(s)$ (resp. $x_{\mathrm{opt}}(s) \leqslant x_{\mathrm{opt}}(s^i_+) \leqslant v_i$) holds.*

Proof. We only prove $v_i \leqslant x_{\mathrm{opt}}(s^i_+) \leqslant x_{\mathrm{opt}}(s)$ for a given integer $i \in [0, n]$ such that $v_0 \leqslant v_i \leqslant x_{\mathrm{opt}}(s)$ (the other case can be treated similarly). We first prove $x_{\mathrm{opt}}(s^i_+) \leqslant x_{\mathrm{opt}}(s)$ by contradiction: suppose that $x_{\mathrm{opt}}(s^i_+) > x_{\mathrm{opt}}(s)$. Let x_{mid} be the midpoint of $x_{\mathrm{opt}}(s^i_+)$ and $x_{\mathrm{opt}}(s)$:

$$x_{\mathrm{mid}} = \frac{x_{\mathrm{opt}}(s^i_+) + x_{\mathrm{opt}}(s)}{2}. \tag{26}$$

Then, by Proposition 1(ii), (iii), we have

$$\Theta_L\left(x_{\mathrm{mid}}, s^i_+\right) < \Theta_R\left(x_{\mathrm{mid}}, s^i_+\right) \quad \text{and} \quad \Theta_R(x_{\mathrm{mid}}, s) < \Theta_L(x_{\mathrm{mid}}, s). \tag{27}$$

Note that by $x_{\mathrm{opt}}(s) < x_{\mathrm{mid}}$ and the assumption of $v_i \leqslant x_{\mathrm{opt}}(s)$, we have $v_i < x_{\mathrm{mid}}$. By (8), $\Theta_R(x_{\mathrm{mid}}, s)$ is the maximum of $f^j_R(x_{\mathrm{mid}}, s)$ for all j such that $v_j > x_{\mathrm{mid}}$, so $\Theta_R(x_{\mathrm{mid}}, s)$ does not change if $w(v_i, s)$ increases to $w^+(v_i)$, i.e.,

$$\Theta_R\left(x_{\mathrm{mid}}, s^i_+\right) = \Theta_R(x_{\mathrm{mid}}, s). \tag{28}$$

Thus, by (27) and (28), we obtain $\Theta_L(x_{\mathrm{mid}}, s^i_+) < \Theta_L(x_{\mathrm{mid}}, s)$ which contradicts Claim 3(i).

We next prove $v_i \leqslant x_{\mathrm{opt}}(s^i_+)$. If $\Theta_L(v_i, s^i_+) \leqslant \Theta_R(v_i, s^i_+)$ holds, $v_i \leqslant x_{\mathrm{opt}}(s^i_+)$ by Lemma 1. We now consider the case of $\Theta_L(v_i, s^i_+) > \Theta_R(v_i, s^i_+)$. By (7) and (8),

$$\Theta_L\left(v_i, s^i_+\right) = \Theta_L(v_i, s) \quad \text{and} \quad \Theta_R\left(v_i, s^i_+\right) = \Theta_R(v_i, s) \tag{29}$$

hold, i.e.,

$$\Theta\left(v_i, s^i_+\right) = \Theta(v_i, s). \tag{30}$$

Thus, we can derive $\Theta_L(v_i, s) > \Theta_R(v_i, s)$ from $\Theta_L(v_i, s^i_+) > \Theta_R(v_i, s^i_+)$, which implies $v_i \geqslant x_{\text{opt}}(s)$ by Lemma 1. By this and the condition $v_i \leqslant x_{\text{opt}}(s)$, we have

$$x_{\text{opt}}(s) = v_i. \tag{31}$$

Here, we show that $x_{\text{opt}}(s^i_+) = v_i$ also holds. Suppose otherwise, i.e., $x_{\text{opt}}(s^i_+) \neq v_i$. Then, we have

$$\Theta\left(x_{\text{opt}}\left(s^i_+\right), s\right) \leqslant \Theta\left(x_{\text{opt}}\left(s^i_+\right), s^i_+\right), \quad \text{and} \tag{32}$$

$$\Theta\left(x_{\text{opt}}\left(s^i_+\right), s^i_+\right) < \Theta\left(v_i, s^i_+\right) \tag{33}$$

by Claim 3(i) and the optimality of $x_{\text{opt}}(s^i_+)$, respectively. From (30), (32), (33) and (31), we can derive $\Theta(x_{\text{opt}}(s^i_+), s) < \Theta(x_{\text{opt}}(s), s)$, which contradicts the optimality of $x_{\text{opt}}(s)$ under s. Therefore, $x_{\text{opt}}(s^i_+) = v_i$ holds if $\Theta_L(v_i, s^i_+) > \Theta_R(v_i, s^i_+)$, which implies that $v_i \leqslant x_{\text{opt}}(s^i_+)$ always holds. □

3. Algorithm

We will show an $O(n \log n)$ time algorithm which computes a minimax regret sink location x^* minimizing a function $R_{\max}(x)$. The algorithm consists of two phases:

[Phase I] Compute $\Theta_{\text{opt}}(s^i_L)$ and $\Theta_{\text{opt}}(s^i_R)$ for all $i \in [0, n]$, and
[Phase II] Compute a minimax regret sink location x^*.

In Phase II, the algorithm applies binary search to find x^* by the unimodality of $R_{\max}(x)$ (recall Claim 2). In order to apply binary search, the algorithm needs to compute $R_{\max}(v_j)$ for any $j \in [0, n]$ by computing the maximum of $\Theta(v_j, s) - \Theta_{\text{opt}}(s)$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$. For this purpose, the algorithm prepares $\Theta_{\text{opt}}(s)$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$ in Phase I. In Sections 3.1 and 3.2, we will explain Phases I and II in detail, respectively.

3.1. Phase I

In this section, we show how to compute $\Theta_{\text{opt}}(s^i_L)$ for all $i \in [0, n]$. Computing $\Theta_{\text{opt}}(s^i_R)$ can be done similarly, and thus is omitted. Now, let us consider how to compute $\Theta_{\text{opt}}(s^i_L)$ for a given $i \in [0, n]$. If $\Theta_L(v_j, s^i_L)$ and $\Theta_R(v_j, s^i_L)$ are computed for some $j \in [0, n]$, the algorithm can determine whether $\Theta_L(v_j, s^i_L) \geqslant \Theta_R(v_j, s^i_L)$ or $\Theta_L(v_j, s^i_L) \leqslant \Theta_R(v_j, s^i_L)$, which implies by Lemma 1 $x_{\text{opt}}(s^i_L) \leqslant v_j$ or $x_{\text{opt}}(s^i_L) \geqslant v_j$, respectively. Therefore, the algorithm can apply binary search to find adjacent two vertices v_{l-1} and v_l with some $l \in [1, n]$ such that $\Theta_L(v_{l-1}, s^i_L) \leqslant \Theta_R(v_{l-1}, s^i_L)$ and $\Theta_L(v_l, s^i_L) \geqslant \Theta_R(v_l, s^i_L)$, i.e., $x_{\text{opt}}(s^i_L)$ exists in the interval $[v_{l-1}, v_l]$. We will explain the details about how to compute $x_{\text{opt}}(s^i_L)$ later.

When $x_{\text{opt}}(s^i_L)$ is found, $\Theta_{\text{opt}}(s^i_L)$ is immediately computed by the definition of $\Theta_{\text{opt}}(s^i_L) = \Theta(x_{\text{opt}}(s^i_L), s^i_L)$. Thus, in order to compute $\Theta_{\text{opt}}(s^i_L)$ for a given $i \in [0, n]$, the algorithm needs to efficiently compute $\Theta_L(v_j, s^i_L)$ and $\Theta_R(v_j, s^i_L)$ for any $j \in [0, n]$. For this purpose, the algorithm constructs two data structures T_L and T_R. We first show that T_L and T_R can be constructed in $O(n \log n)$ time and $\Theta_L(v_j, s^i_L)$ or $\Theta_R(v_j, s^i_L)$ for any $i, j \in [0, n]$ can be computed in $O(\log n)$ time by using T_L and T_R, respectively. Here, we explain T_L in detail (T_R is can be constructed in a symmetric manner). Let us consider a priority search tree with $n + 1$ leaves $l_0, l_1, \ldots, l_n$ corresponding to vertices $v_0, v_1, \ldots, v_n$ and internal nodes such that each internal node has pointers to left and right children. For a node ν in a priority search tree, let $\kappa_L(\nu)$ (resp. $\kappa_R(\nu)$) denote the left (resp. right) child of ν, and $i_{\min}(\nu)$ (resp. $i_{\max}(\nu)$) denote the index of a minimum (resp. maximum) leaf of a subtree rooted at ν, which are stored at ν. Note that for a leaf l_j, $i_{\min}(l_j) = i_{\max}(l_j) = j$ holds. For $i \in [0, n]$, let T^i_L denote a priority search tree such that each node (including leaf) ν stores an interval $[i_{\min}(\nu), i_{\max}(\nu)]$,

$$\mathit{value}(\nu; i) = \max\left\{ -v_k\tau + \sum_{l \in [0,k]} w\left(v_l, s^i_L\right) \;\middle|\; k \in \left[i_{\min}(\nu), i_{\max}(\nu)\right] \right\} \tag{34}$$

and the corresponding index of the leaf that attains the maximum. Then, for any vertex v_j with $j \in [0, n]$, by (5) and (34), we have

$$v_j\tau + \mathit{value}(\nu; i) = \max\left\{ f^k_L\left(v_j, s^i_L\right) \mid k \in \left[i_{\min}(\nu), i_{\max}(\nu)\right] \right\}. \tag{35}$$

Also, for $i \in [0, n]$, let Π^i_L be a data structure along the path in T^i_L from leaf l_i to the root (see Fig. 5). Instead of keeping the whole tree T^i_L for $i \in [1, n]$, we only store Π^i_L. This is enough for our purpose. Thus, T_L basically consists of a priority

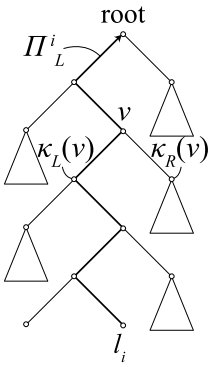

Fig. 5. Illustration of Π_L^j.

search tree T_L^0 and path data structures Π_L^i for $i \in [1,n]$, i.e., the algorithm first constructs T_L^0 and subsequently constructs $\Pi_L^1, \Pi_L^2, \ldots, \Pi_L^n$. We have the following claim.

Claim 4. *T_L and T_R can be constructed in $O(n \log n)$ time and $O(n \log n)$ space.*

Proof. We only prove for T_L. First, we notice that T_L^0 can be constructed in $O(n)$ time and $O(n)$ space, and then, Π_L^0 can be constructed in $O(\log n)$ time and $O(\log n)$ space. Suppose that $\Pi_L^0, \ldots, \Pi_L^{i-1}$ for a given $i \in [1,n]$ have been constructed. The algorithm then follows the path Π_L^i from leaf l_i to the root and stores $value(v; i)$ at each node v on Π_L^i, which takes $O(\log n)$ time and $O(\log n)$ space. At leaf l_i, the algorithm sets

$$value(l_i; i) = -v_i\tau + \sum_{j \in [0,i]} w\big(v_j, s_L^i\big) \tag{36}$$

$$= value(l_i; 0) + \sum_{j \in [1,i]} \big(w^+(v_j) - w^-(v_j)\big). \tag{37}$$

Here, suppose that the algorithm has precomputed $\sum_{j \in [1,i]}(w^+(v_j) - w^-(v_j))$ for all $i \in [1,n]$ in $O(n)$ time and $O(n)$ space. Suppose that for an internal node v on Π_L^i, the algorithm has already set $value(v'; i)$ for every node v' on Π_L^i between l_i and v. Then, the algorithm sets $value(v; i)$ as the maximum of $value(\kappa_R(v); i)$ and $value(\kappa_L(v); i)$. If $\kappa_L(v)$ is on Π_L^i, since $value(\kappa_L(v); i)$ has already been computed, the algorithm only computes $value(\kappa_R(v); i)$ as

$$value\big(\kappa_R(v); i\big) = value\big(\kappa_R(v); 0\big) + \sum_{j \in [1,i]} \big(w^+(v_j) - w^-(v_j)\big). \tag{38}$$

If $\kappa_R(v)$ is on Π_L^i, since $value(\kappa_R(v); i)$ has already been computed, the algorithm only needs to obtain $value(\kappa_L(v); i)$. We notice that $i_{\max}(\kappa_L(v)) < i$ holds, and then, for any $l \in [0, i_{\max}(\kappa_L(v))]$, $w(v_l, s_L^i) = w(v_l, s_L^{i_{\max}(\kappa_L(v))}) = w^+(v_l)$ holds by (16), which implies

$$value\big(\kappa_L(v); i\big) = value\big(\kappa_L(v); i_{\max}\big(\kappa_L(v)\big)\big). \tag{39}$$

This can be derived from $\Pi_L^{i_{\max}(\kappa_L(v))}$ (which has already been obtained). So, $\Pi_L^1, \Pi_L^2, \ldots, \Pi_L^n$ can be constructed in $O(n \log n)$ time and $O(n \log n)$ space. □

We now show how to compute $\Theta_L(v_j, s_L^i)$ for integers $i, j \in [0,n]$ by using T_L (recall that we assume $\Theta_L(v_0, s) = 0$ and $\Theta_R(v_n, s) = 0$ for any scenario s). By (7), we have

$$\Theta_L\big(v_j, s_L^i\big) = \max\big\{ f_L^k\big(v_j, s_L^i\big) \,\big|\, k \in [0, j-1] \big\}. \tag{40}$$

Note that $\bigcup\{[i_{\min}(\kappa_L(v)), i_{\max}(\kappa_L(v))] \mid v \text{ on } \Pi_L^j\} = [0, j-1]$. Thus, by (35) and (40), $\Theta_L(v_j, s_L^i)$ can be represented as

$$\Theta_L\big(v_j, s_L^i\big) = v_j\tau + \max\big\{ value\big(\kappa_L(v); i\big) \,\big|\, v \text{ on } \Pi_L^j \big\}. \tag{41}$$

So, the algorithm is actually required to compute the maximum of $value(\kappa_L(v); i)$ for all nodes v on Π_L^j. There are two cases:

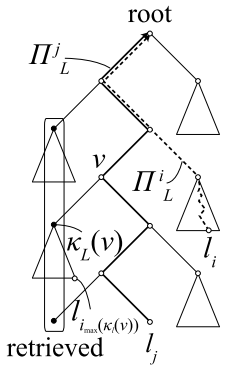

Fig. 6. Illustration of Case 1.

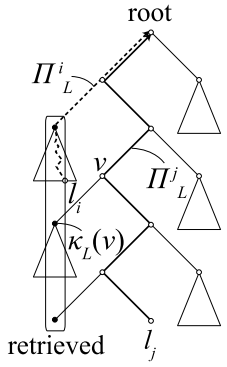

Fig. 7. Illustration of Case 2.

[Case 1] $j \leqslant i$ (see Fig. 6), and
[Case 2] $j > i$ (see Fig. 7).

[Case 1]: The algorithm follows the path Π^j_L from leaf l_j to the root. Every time the algorithm visits an internal node v, it examines whether $\kappa_L(v)$ is on Π^j_L or not. If this is the case, the algorithm does nothing. Otherwise, the algorithm needs to obtain $value(\kappa_L(v); i)$. Recall that $i_{\max}(\kappa_L(v)) < i$ holds, and then, for any $l \in [0, i_{\max}(\kappa_L(v))]$, $w(v_l, s^i_L) = w(v_l, s^{i_{\max}(\kappa_L(v))}_L) = w^+(v_l)$ holds by (16), which implies

$$value\big(\kappa_L(v); i\big) = value\big(\kappa_L(v); i_{\max}\big(\kappa_L(v)\big)\big). \tag{42}$$

So, the algorithm gets the rightmost leaf $l_{i_{\max}(\kappa_L(v))}$ in the subtree rooted at $\kappa_L(v)$, and retrieves $value(\kappa_L(v); i_{\max}(\kappa_L(v)))$ stored in $\Pi^{i_{\max}(\kappa_L(v))}_L$. The algorithm continues to do this computation and to take the maximum value among those retrieved, which takes $O(\log n)$ time.

[Case 2]: The task the algorithm does is similar to Case 1. Every time the algorithm visits an internal node v before Π^j_L encounters the node on Π^i_L, it examines whether $\kappa_L(v)$ is on Π^j_L or not. If this is the case, the algorithm does nothing. Otherwise, the algorithm retrieves $value(\kappa_L(v); 0)$ stored in T^0_L and adds $\sum_{j\in[1,i]}(w^+(v_j) - w^-(v_j))$ to the retrieved value since

$$value\big(\kappa_L(v); i\big) = value\big(\kappa_L(v); 0\big) + \sum_{j\in[1,i]} \big(w^+(v_j) - w^-(v_j)\big) \tag{43}$$

holds. Here, suppose that the algorithm has precomputed $\sum_{j\in[1,i]}(w^+(v_j) - w^-(v_j))$ for all $i \in [1, n]$ in $O(n)$ time. The algorithm continues to do this computation and to take the maximum value among those retrieved before encountering the node on Π^i_L, and after that, it does the same computation as in Case 1, which takes $O(\log n)$ time.

We then have the following claim.

Claim 5. *For any integers $i, j \in [0, n]$, $\Theta_L(v_j, s^i_L)$ and $\Theta_R(v_j, s^i_L)$ can be computed in $O(\log n)$ time once T_L and T_R have been constructed.*

Next, we show how the algorithm actually computes $\Theta_{\mathrm{opt}}(s_L^i)$ for all $i \in [0, n]$. The algorithm first computes $\Theta_{\mathrm{opt}}(s_L^0)$ and $x_{\mathrm{opt}}(s_L^0)$. Based on Lemma 1, the algorithm can apply binary search to find adjacent two vertices v_{l-1} and v_l with some $l \in [1, n]$ such that $\Theta_L(v_{l-1}, s_L^0) \leqslant \Theta_R(v_{l-1}, s_L^0)$ and $\Theta_L(v_l, s_L^0) \geqslant \Theta_R(v_l, s_L^0)$, i.e., $x_{\mathrm{opt}}(s_L^0)$ exists in the interval $[v_{l-1}, v_l]$, which takes $O(\log^2 n)$ time by Claim 5. We now show how to find $x_{\mathrm{opt}}(s_L^0)$ from $[v_{l-1}, v_l]$ in constant time. Let $x(t)$ denote a point dividing $[v_{l-1}, v_l]$ with the ratio of t to $1-t$. Suppose that $0 < t < 1$. Then, by the conditions $\Theta(v_l, s_L^0) = \Theta_L(v_l, s_L^0)$ and $\Theta(v_{l-1}, s_L^0) = \Theta_R(v_{l-1}, s_L^0)$, we have

$$\Theta\big(x(t), s_L^0\big) = \max\big\{\Theta\big(v_l, s_L^0\big) - (1-t)\,(v_l - v_{l-1})\tau,\, \Theta\big(v_{l-1}, s_L^0\big) - t(v_l - v_{l-1})\tau\big\}. \tag{44}$$

Let us consider the solution t^* of $\Theta(v_l, s_L^0) - (1-t)(v_l - v_{l-1})\tau = \Theta(v_{l-1}, s_L^0) - t(v_l - v_{l-1})\tau$. If $0 < t^* < 1$ holds, $x_{\mathrm{opt}}(s_L^0) = x(t^*)$ holds by Lemma 1. If $t^* \leqslant 0$ holds, then $\Theta(v_{l-1}, s_L^0) \leqslant \Theta(v_l, s_L^0) - (v_l - v_{l-1})\tau$ holds, which implies $x_{\mathrm{opt}}(s_L^0) = v_{l-1}$. Similarly, if $t^* \geqslant 1$ holds, $x_{\mathrm{opt}}(s_L^0) = v_l$ holds. Therefore, the algorithm can compute $\Theta_{\mathrm{opt}}(s_L^0)$ and $x_{\mathrm{opt}}(s_L^0)$ in $O(\log^2 n)$ time. Generally, we have the following claim.

Claim 6. *For a scenario $s \in \mathcal{S}$, suppose that $\Theta_L(v_{l-1}, s) \leqslant \Theta_R(v_{l-1}, s)$ and $\Theta_L(v_l, s) \geqslant \Theta_R(v_l, s)$ hold for adjacent two vertices v_{l-1} and v_l with some $l \in [1, n]$, and let t^* denote the solution to an equation $\Theta(v_{l-1}, s) - t(v_l - v_{l-1})\tau = \Theta(v_l, s) - (1-t)(v_l - v_{l-1})\tau$. Then,*

(i) *if $0 \leqslant t^* \leqslant 1$ holds, $x_{\mathrm{opt}}(s)$ is a point dividing the interval $[v_{l-1}, v_l]$ with the ratio of t^* to $1-t^*$ and $\Theta_{\mathrm{opt}}(s) = \Theta_R(v_{l-1}, s) - t^*(v_l - v_{l-1})\tau$ holds,*
(ii) *if $t^* < 0$ holds, $x_{\mathrm{opt}}(s) = v_{l-1}$ and $\Theta_{\mathrm{opt}}(s) = \Theta_R(v_{l-1}, s)$ hold, and*
(iii) *if $t^* > 1$ holds, $x_{\mathrm{opt}}(s) = v_l$ and $\Theta_{\mathrm{opt}}(s) = \Theta_L(v_l, s)$ hold.*

$\Theta_{\mathrm{opt}}(s_L^i)$ and $x_{\mathrm{opt}}(s_L^i)$ for every i can be computed in $O(\log^2 n)$ in the same manner as $\Theta_{\mathrm{opt}}(s_L^0)$ and $x_{\mathrm{opt}}(s_L^0)$. Thus, $\Theta_{\mathrm{opt}}(s_L^i)$ and $x_{\mathrm{opt}}(s_L^i)$ for all $i \in [0, n]$ can be obtained in $O(n \log^2 n)$ time. This running time is further improved to $O(n \log n)$ time based on the nontrivial observation stated below. We first have the following claim by Lemma 4.

Claim 7. *For an integer $i \in [0, n-1]$,*

(i) *if $x_{\mathrm{opt}}(s_L^i) \geqslant v_{i+1}$ holds, $v_{i+1} \leqslant x_{\mathrm{opt}}(s_L^{i+1}) \leqslant x_{\mathrm{opt}}(s_L^i)$ holds, and*
(ii) *if $x_{\mathrm{opt}}(s_L^i) \leqslant v_{i+1}$ holds, $x_{\mathrm{opt}}(s_L^i) \leqslant x_{\mathrm{opt}}(s_L^{i+1}) \leqslant v_{i+1}$ holds.*

From this we immediately have the following claim.

Claim 8. (i) *As long as $x_{\mathrm{opt}}(s_L^i) \geqslant v_{i+1}$ holds, $x_{\mathrm{opt}}(s_L^i)$ does not increase as i increases.* (ii) *Once $x_{\mathrm{opt}}(s_L^i) \leqslant v_{i+1}$ holds, $x_{\mathrm{opt}}(s_L^i)$ never decreases as i increases.*

For every $i \in [0, n]$, let $l(i)$ denote an integer such that $\Theta_L(v_{l(i)-1}, s_L^i) \leqslant \Theta_R(v_{l(i)-1}, s_L^i)$ and $\Theta_L(v_{l(i)}, s_L^i) \geqslant \Theta_R(v_{l(i)}, s_L^i)$ hold, i.e., $x_{\mathrm{opt}}(s_L^i)$ exists in the interval $[v_{l(i)-1}, v_{l(i)}]$. Suppose that the algorithm has already computed $x_{\mathrm{opt}}(s_L^i)$ and $[v_{l(i)-1}, v_{l(i)}]$. Then, based on Claim 7, the improved procedure which finds $[v_{l(i+1)-1}, v_{l(i+1)}]$ is described as follows.

(a) If the condition of Claim 7(i) holds, $x_{\mathrm{opt}}(s_L^{i+1})$ lies to the left of $v_{l(i)}$. The algorithm first tests the interval $[v_{l(i)-1}, v_{l(i)}]$ whether $\Theta_L(v_{l(i)-1}, s_L^{i+1}) \leqslant \Theta_R(v_{l(i)-1}, s_L^{i+1})$ and $\Theta_L(v_{l(i)}, s_L^{i+1}) \geqslant \Theta_R(v_{l(i)}, s_L^{i+1})$ hold or not. If this is the case, $x_{\mathrm{opt}}(s_L^{i+1})$ exists in $[v_{l(i)-1}, v_{l(i)}]$. Otherwise, the algorithm sequentially tests $[v_{l(i)-2}, v_{l(i)-1}], [v_{l(i)-3}, v_{l(i)-2}], \ldots$ in this order to locate $[v_{l(i+1)-1}, v_{l(i+1)}]$ where $x_{\mathrm{opt}}(s_L^{i+1})$ exists.

(b) If the condition of Claim 7(ii) holds, $x_{\mathrm{opt}}(s_L^{i+1})$ lies to the right of $v_{l(i)-1}$. The algorithm first tests the interval $[v_{l(i)-1}, v_{l(i)}]$ whether $\Theta_L(v_{l(i)-1}, s_L^{i+1}) \leqslant \Theta_R(v_{l(i)-1}, s_L^{i+1})$ and $\Theta_L(v_{l(i)}, s_L^{i+1}) \geqslant \Theta_R(v_{l(i)}, s_L^{i+1})$ hold or not. If this is the case, $x_{\mathrm{opt}}(s_L^{i+1})$ exists in $[v_{l(i)-1}, v_{l(i)}]$. Otherwise, the algorithm sequentially tests $[v_{l(i)}, v_{l(i)+1}], [v_{l(i)+1}, v_{l(i)+2}], \ldots$ in this order to locate $[v_{l(i+1)-1}, v_{l(i+1)}]$ where $x_{\mathrm{opt}}(s_L^{i+1})$ exists.

We now analyze the running time of this algorithm. Here, we recall Lemma 1. For given i and j, the algorithm tests whether $x_{\mathrm{opt}}(s_L^i)$ exists in $[v_{j-1}, v_j]$ or not by computing $\Theta_L(v_{j-1}, s_L^i), \Theta_R(v_{j-1}, s_L^i), \Theta_L(v_j, s_L^i)$ and $\Theta_R(v_j, s_L^i)$. This computation takes $O(\log n)$ time by Claim 5. And by Claim 8, there exists an integer $i' \in [0, n]$ such that $x_{\mathrm{opt}}(s_L^i) \geqslant v_{i+1}$ holds for every $i < i'$ and $x_{\mathrm{opt}}(s_L^{i'}) \leqslant v_{i'+1}$ holds. Then, we have $l(i') \leqslant l(i'-1) \leqslant \cdots \leqslant l(1) \leqslant l(0)$ and $l(i') \leqslant l(i'+1) \leqslant \cdots \leqslant l(n-1) \leqslant l(n)$. For a given $i < i'$, the algorithm tests $l(i) - l(i+1) + 1$ intervals, i.e., $O(n)$ intervals are tested for all $i < i'$ in total. Similarly, $O(n)$ intervals are tested for all $i \geqslant i'$. Therefore, the overall running time is $O(n \log n)$.

Claim 9. *$\Theta_{\mathrm{opt}}(s_L^i)$ for all $i \in [0, n]$ can be computed in $O(n \log n)$ time once T_L and T_R have been constructed.*

By Claims 4 and 9, $\Theta_{\mathrm{opt}}(s_L^i)$ for all $i \in [0,n]$ can be computed in $O(n \log n)$ time and $O(n \log n)$ space, and $\Theta_{\mathrm{opt}}(s_R^i)$ can be treated similarly.

Lemma 5. *$\Theta_{\mathrm{opt}}(s_L^i)$ and $\Theta_{\mathrm{opt}}(s_R^i)$ for all $i \in [0,n]$ can be computed in $O(n \log n)$ time and $O(n \log n)$ space.*

3.2. Phase II

In this section, we show how to compute a minimax regret sink location x^*. Recall that binary search can be applied to find x^* by the unimodality of $R_{\max}(x)$. Actually, the algorithm finds the interval $[v_{l-1}, v_l]$ where x^* exists based on Lemma 2. For this purpose, the algorithm needs to compute a worst case scenario for v_j with any $j \in [0,n]$ as

$$\hat{s}_j = \operatorname{argmax}\left\{ R(v_j, s) \;\middle|\; s \in \mathcal{S}_L \cup \mathcal{S}_R \right\} \tag{45}$$

by evaluating $R(v_j, s)$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$. By the definition (11), we have $R(v_j, s) = \Theta(v_j, s) - \Theta_{\mathrm{opt}}(s)$. In Phase I, $\Theta_{\mathrm{opt}}(s)$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$ have been computed. Therefore, in Phase II, the algorithm only needs to compute $\Theta(v_j, s)$ for each $s \in \mathcal{S}_L \cup \mathcal{S}_R$, and then, $R(v_j, s)$ can be immediately computed. Once computing $\hat{s}_j$, the algorithm can know whether $x^* \leqslant v_j$ or $x^* \geqslant v_j$ by evaluating $\Theta_L(v_j, \hat{s}_j)$ and $\Theta_R(v_j, \hat{s}_j)$ by Lemma 2.

We first show that the algorithm can compute $\Theta(v_j, s)$ for a given $j \in [0,n]$ and all $s \in \mathcal{S}_L \cup \mathcal{S}_R$ in $O(n)$ time in total. The algorithm basically needs to compute $\Theta_L(v_j, s_L^i)$, $\Theta_R(v_j, s_L^i)$, $\Theta_L(v_j, s_R^i)$ and $\Theta_R(v_j, s_R^i)$ for all $i \in [0,n]$. In the following, we only show how to compute $\Theta_L(v_j, s_L^i)$ for all $i \in [0, j-1]$ in $O(n)$ time in total (for $i \in [j,n]$, $\Theta_L(v_j, s_L^i) = \Theta_L(v_j, s_L^{j-1})$ holds by the definitions (2) and (16)). Note that the other cases can be treated similarly.

For $i \in [0, j-1]$, let $id_{\max}(i)$ denote the integer defined as

$$id_{\max}(i) = \operatorname{argmax}\left\{ -v_k\tau + \sum_{l \in [0,k]} w\left(v_l, s_L^i\right) \;\middle|\; k \in [0, j-1] \right\}, \tag{46}$$

i.e., $\Theta_L(v_j, s_L^i) = f_L^{id_{\max}(i)}(v_j, s_L^i)$ by (5) and (7). The algorithm first computes $\Theta_L(v_j, s_L^0)$ and $id_{\max}(0)$ by using T_L (as mentioned in Section 3.1), which can be done in $O(\log n)$ time by Claim 5. Here, suppose that the algorithm has already obtained $\Theta_L(v_j, s_L^i)$ and $id_{\max}(i)$ for a given $i \in [0, n-1]$. We then show that the algorithm can compute $\Theta_L(v_j, s_L^{i+1})$ and $id_{\max}(i+1)$ in constant time. There are two cases:

[Case 1] $id_{\max}(i) \geqslant i+1$, and
[Case 2] $id_{\max}(i) < i+1$.

In the following, let $\Delta_i = w^+(v_i) - w^-(v_i)$.

[Case 1]: In this case, $id_{\max}(i+1)$ does not change from $id_{\max}(i)$, which implies $\Theta_L(v_j, s_L^{i+1}) = f_L^{id_{\max}(i)}(v_j, s_L^{i+1}) = f_L^{id_{\max}(i)}(v_j, s_L^i) + \Delta_{i+1}$ (see Fig. 8). We have $f_L^{id_{\max}(i)}(v_j, s_L^i) = \Theta_L(v_j, s_L^i)$. Thus, $\Theta_L(v_j, s_L^{i+1})$ and $id_{\max}(i+1)$ can be computed as

$$\Theta_L\left(v_j, s_L^{i+1}\right) = \Theta_L\left(v_j, s_L^i\right) + \Delta_{i+1}, \quad \text{and} \tag{47}$$

$$id_{\max}(i+1) = id_{\max}(i). \tag{48}$$

[Case 2]: In this case, for $k \in [id_{\max}(i), i]$, $f_L^k(v_j, s_L^{i+1}) = f_L^k(v_j, s_L^i)$ holds, and for $k \in [i+1, j-1]$, $f_L^k(v_j, s_L^{i+1}) = f_L^k(v_j, s_L^i) + \Delta_{i+1}$ holds (see Fig. 9). For $i \in [0, j-2]$, we define $V(i)$ as

$$V(i) = \max\left\{ -v_k\tau + \sum_{l \in [0,k]} w\left(v_l, s_L^i\right) \;\middle|\; k \in [i+1, j-1] \right\} \tag{49}$$

$$= \max\left\{ -v_k\tau + \sum_{l \in [0,k]} w\left(v_l, s_L^0\right) \;\middle|\; k \in [i+1, j-1] \right\} + \sum_{h \in [1,i]} \Delta_h, \tag{50}$$

and let $\widetilde{id}_{\max}(i)$ denote the integer which attains the maximum in (50). When an integer j is given, $V(i)$ and $\widetilde{id}_{\max}(i)$ for all $i \in [0, j-2]$ can be precomputed in $O(n)$ time as follows. The first term of (50) for all $i \in [0, j-2]$ can be computed in $O(n)$ time in total by computing them in descending order of i, and also, the second term of (50) for all $i \in [0, j-2]$ can be computed in $O(n)$ time in total by computing them in ascending order of i. Then,

$$\Theta_L\left(v_j, s_L^{i+1}\right) = \max\left\{ f_L^{id_{\max}(i)}\left(v_j, s_L^{i+1}\right), f_L^{\widetilde{id}_{\max}(i)}\left(v_j, s_L^{i+1}\right) \right\} \tag{51}$$

$$= \max\left\{ f_L^{id_{\max}(i)}\left(v_j, s_L^i\right), f_L^{\widetilde{id}_{\max}(i)}\left(v_j, s_L^i\right) + \Delta_{i+1} \right\} \tag{52}$$

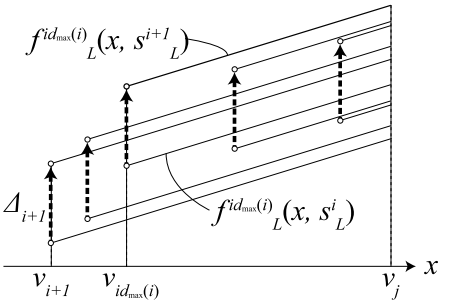

Fig. 8. Illustration of Case 1.

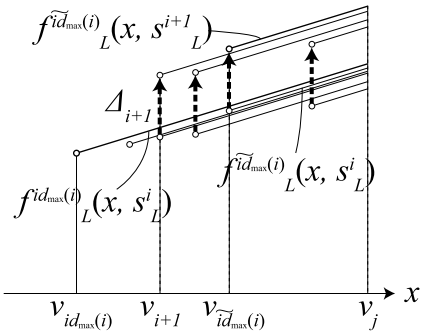

Fig. 9. Illustration of Case 2.

holds. We have $f_L^{id_{\max}(i)}(v_j, s_L^i) = \Theta_L(v_j, s_L^i)$ and $f_L^{\widetilde{id}_{\max}(i)}(v_j, s_L^i) = v_j\tau + V(i)$. By these and (52), once $V(i)$ and $\widetilde{id}_{\max}(i)$ are computed for all $i \in [0, j-2]$ in advance, $\Theta_L(v_j, s_L^{i+1})$ and $id_{\max}(i+1)$ can be computed as

$$\Theta_L\big(v_j, s_L^{i+1}\big) = \max\big\{\Theta_L\big(v_j, s_L^i\big), v_j\tau + V(i) + \Delta_{i+1}\big\}, \quad \text{and} \tag{53}$$

$$id_{\max}(i+1) = \begin{cases} id_{\max}(i) & \text{if } \Theta_L(v_j, s_L^i) \geqslant v_j\tau + V(i) + \Delta_{i+1}, \\ \widetilde{id}_{\max}(i) & \text{if } \Theta_L(v_j, s_L^i) < v_j\tau + V(i) + \Delta_{i+1}. \end{cases} \tag{54}$$

In the above mentioned manner, the algorithm can compute $\Theta_L(v_j, s_L^i)$ for all $i \in [0, j-1]$ in $O(n)$ time. Since $\Theta_R(v_j, s_L^i)$, $\Theta_L(v_j, s_R^i)$ and $\Theta_R(v_j, s_R^i)$ can be similarly computed in $O(n)$ time, we have the following claim.

Claim 10. *For a given $j \in [0, n]$, let $\hat{s}_j = \operatorname{argmax}\{R(v_j, s) \mid s \in \mathcal{S}_L \cup \mathcal{S}_R\}$. Then, $\hat{s}_j$ can be computed in $O(n)$ time once $\Theta_{\text{opt}}(s_L^i)$ and $\Theta_{\text{opt}}(s_R^i)$ for all $i \in [0, n]$ have been computed.*

When the algorithm computes $\hat{s}_j$ for a given $j \in [0, n]$, it can know whether $x^* \leqslant v_j$ or $x^* \geqslant v_j$ by Lemma 2. Therefore, the algorithm can apply binary search to find the interval $[v_{l-1}, v_l]$ with some $l \in [1, n]$ where x^* exists such that $\Theta_L(v_{l-1}, \hat{s}_{l-1}) \leqslant \Theta_R(v_{l-1}, \hat{s}_{l-1})$ and $\Theta_L(v_l, \hat{s}_l) \geqslant \Theta_R(v_l, \hat{s}_l)$, which takes $O(n \log n)$ time by Claim 10. We now show how to find x^* from $[v_{l-1}, v_l]$ in constant time, which is similar to the manner discussed at Claim 6. Let $x(t)$ denote a point dividing the interval $[v_{l-1}, v_l]$ with the ratio of t to $1-t$. Suppose that $0 < t < 1$. Then, by the conditions $R_{\max}(v_l) = \Theta_L(v_l, \hat{s}_l) - \Theta_{\text{opt}}(\hat{s}_l)$ and $R_{\max}(v_{l-1}) = \Theta_R(v_{l-1}, \hat{s}_{l-1}) - \Theta_{\text{opt}}(\hat{s}_{l-1})$, we have

$$R_{\max}\big(x(t)\big) = \max\big\{R_{\max}(v_l) - (1-t)\,(v_l - v_{l-1})\tau\,,\, R_{\max}(v_{l-1}) - t(v_l - v_{l-1})\tau\big\}. \tag{55}$$

Let us consider the solution t^* of $R_{\max}(v_l) - (1-t)(v_l - v_{l-1})\tau = R_{\max}(v_{l-1}) - t(v_l - v_{l-1})\tau$. If $0 < t^* < 1$ holds, $x^* = x(t^*)$ holds by Lemma 2. If $t^* \leqslant 0$ holds, then $R_{\max}(v_{l-1}) \leqslant R_{\max}(v_l) - (v_l - v_{l-1})\tau$ holds, which implies $x^* = v_{l-1}$. Similarly, if $t^* \geqslant 1$ holds, $x^* = v_l$ holds. We then have the following lemma.

Lemma 6. *A minimax regret sink location x^* can be computed in $O(n \log n)$ time once $\Theta_{\text{opt}}(s_L^i)$ and $\Theta_{\text{opt}}(s_R^i)$ for all $i \in [0, n]$ have been computed.*

By Lemmas 5 and 6, we obtain the following theorem.

Theorem 1. *The minimax regret* 1*-sink location problem in a dynamic path network with uniform capacity can be solved in* $O(n \log n)$ *time and* $O(n \log n)$ *space.*

4. Conclusion

In this paper, we develop an $O(n \log n)$ time algorithm for the minimax regret 1-sink location problem in dynamic path networks with uniform capacity. Note that recently, Wang [14] independently studied the same problem as ours and proposed an $O(n \log n)$ time and $O(n)$ space algorithm, which relies on some observations given in the preliminary version of this paper [5].

Here recall that each input value is given as an integer and each function always returns an integer in this paper, which is called *discrete model*. On the other hand, if each input value is given as a real number and ceiling is removed from each function, the model is called *continuous model*. Indeed we have solved the problem in discrete model with $c = 1$, the case of $c \geqslant 2$ is still left open. In discrete model with $c \geqslant 2$, there exists a sink location for which any worst case scenario is not left-dominant nor right-dominant, which implies that the algorithm has to consider more than $O(n)$ scenarios. However, in continuous model, all of claims, lemmas and main theorem in this paper are still maintained even if $c \neq 1$, therefore we can directly apply our developed algorithm to continuous model without increasing time complexity and space complexity. Also, we can prove that the solution for continuous model can be regarded as the approximation for discrete model such that the difference between the approximate cost and the optimal cost is at most 1 (details are omitted).

References

[1] I. Averbakh, O. Berman, Algorithms for the robust 1-center problem on a tree, Eur. J. Oper. Res. 123 (2) (2000) 292–302.
[2] B. Bhattacharya, T. Kameda, A linear time algorithm for computing minmax regret 1-median on a tree, in: Proc. COCOON 2012, in: Lect. Notes Comput. Sci., vol. 7434, 2012, pp. 1–12.
[3] G.S. Brodal, L. Georgiadis, I. Katriel, An $O(n \log n)$ version of the Averbakh–Berman algorithm for the robust median of a tree, Oper. Res. Lett. 36 (1) (2008) 14–18.
[4] B. Chen, C. Lin, Minmax-regret robust 1-median location on a tree, Networks 31 (2) (1998) 93–103.
[5] S. Cheng, Y. Higashikawa, N. Katoh, G. Ni, B. Su, Y. Xu, Minimax regret 1-sink location problems in dynamic path networks, in: Proc. 10th Annual Conference on Theory and Applications of Models of Computation (TAMC 2013), in: Lect. Notes Comput. Sci., vol. 7876, 2013, pp. 121–132.
[6] E. Conde, Minmax regret location–allocation problem on a network under uncertainty, Eur. J. Oper. Res. 179 (3) (2007) 1025–1039.
[7] E. Conde, A note on the minmax regret centdian location on trees, Oper. Res. Lett. 36 (2) (2008) 271–275.
[8] L.R. Ford Jr., D.R. Fulkerson, Constructing maximal dynamic flows from static flows, Oper. Res. 6 (1958) 419–433.
[9] N. Kamiyama, N. Katoh, A. Takizawa, An efficient algorithm for evacuation problem in dynamic network flows with uniform arc capacity, IEICE Trans. 89-D (8) (2006) 2372–2379.
[10] P. Kouvelis, G. Yu, Robust Discrete Optimization and Its Applications, Kluwer Academic Publishers, Dordrecht, 1997.
[11] S. Mamada, T. Uno, K. Makino, S. Fujishige, An $O(n \log^2 n)$ algorithm for the optimal sink location problem in dynamic tree networks, Discrete Appl. Math. 154 (16) (2006) 2387–2401.
[12] W. Ogryczak, Conditional median as a robust solution concept for uncapacitated location problems, TOP 18 (1) (2010) 271–285.
[13] J. Puerto, A.M. Rodríguez-Chía, A. Tamir, Minimax regret single-facility ordered median location problems on networks, INFORMS J. Comput. 21 (1) (2009) 77–87.
[14] H. Wang, Minmax regret 1-facility location on uncertain path networks, in: Proc. 24th International Symposium on Algorithms and Computation (ISAAC 2013), in: Lect. Notes Comput. Sci., vol. 8283, 2013, pp. 733–743.