An $O(n^2 \log^2 n)$ Time Algorithm for Minmax Regret Minsum Sink on Path Networks

Binay Bhattacharya
School of Computing Science, Simon Fraser University, Burnaby, Canada

Yuya Higashikawa
School of Business Administration, University of Hyogo, Kobe, Japan

Tsunehiko Kameda
School of Computing Science, Simon Fraser University, Burnaby, Canada

Naoki Katoh
School of Science and Technology, Kwansei Gakuin University, Sanda, Japan

Abstract

We model evacuation in emergency situations by dynamic flow in a network. We want to minimize the aggregate evacuation time to an evacuation center (called a sink) on a path network with uniform edge capacities. The evacuees are initially located at the vertices, but their precise numbers are unknown, and are given by upper and lower bounds. Under this assumption, we compute a sink location that minimizes the maximum "regret." We present the first sub-cubic time algorithm in n to solve this problem, where n is the number of vertices. Although we cast our problem as evacuation, our result is accurate if the "evacuees" are fluid-like continuous material, but is a good approximation for discrete evacuees.

2012 ACM Subject Classification Networks → Network algorithms, Mathematics of computing → Graph algorithms, Applied computing → Transportation

Keywords and phrases Facility location, minsum sink, evacuation problem, minmax regret, dynamic flow path network

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.14

Related Version A full version of the paper is available at [3], `https://arxiv.org/abs/1806.00814`.

Acknowledgements This work is supported in part by NSERC of Canada Discovery Grant, in part by JST CREST (JPMJCR1402), and in part by JSPS KAKENHI Grant-in-Aid for Young Scientists (B) (17K12641).

1 Introduction

The goal of evacuation planning is to evacuate all the evacuees to some sinks, optimizing a certain objective function [8, 16]. Some aspects of such planning can be modeled by dynamic flow in a network [6] whose vertices represent the places where the evacuees are initially located and the edges represent possible evacuation routes. Associated with each edge is the transit time across the edge and its capacity in terms of the number of people who can enter it per unit time. Evacuation starts from all vertices at the same time.

A *completion time k-sink*, a.k.a. *minmax k-sink*, is a set of k sinks that minimizes the time until every evacuee has moved to a sink. If the edge capacities are uniform, it is easy to compute a completion time 1-sink in path networks in linear time [5, 10]. Mamada et al. [16]

solved this problem for the tree networks with non-uniform edge capacities in $O(n \log^2 n)$ time, when the sink is constrained to be at a vertex. Higashikawa et al. proposed an $O(n \log n)$ algorithm without this constraint when the edges have the same capacity [12].

The concept of *regret* was introduced by Kouvelis and Yu [15], to model the situations where optimization is required when the exact values (such as the number of evacuees at the vertices) are unknown, but are given by upper and lower bounds. A particular instance of the set of such numbers, one for each vertex, is called a *scenario*. The objective is to find a solution which is as good as any other solution in the worst case, where the actual scenario is the most unfavorable. Cheng et al. [5] proposed an $O(n \log^2 n)$ time algorithm for finding a minmax regret 1-sink in path networks with uniform edge capacities. This initial result was soon improved to $O(n \log n)$ [10, 17], and further to $O(n)$ [4]. Bhattacharya and Kameda [4] propose an $O(n \log^4 n)$ time algorithm to find a minmax regret 2-sink on path networks. For the k-sink version of the problem, Arumugam et al. [1] give two algorithms, which run in $O(kn^3 \log n)$ and $O(kn^2(\log n)^k)$ time, respectively. As for the tree networks with uniform edge capacities, Higashikawa et al. [12] propose an $O(n^2 \log^2 n)$ time algorithm for finding a minmax regret 1-sink. Golin and Sandeep [7] recently proposed an $O(\max\{k^2, \log^2 n\}k^2n^2 \log^5 n)$ time algorithm for finding a minmax reget k-sink.

The objective function we adopt in this paper is the *aggregate evacuation time*, i.e., the sum of the evacuation time of every evacuee, a.k.a. *minsum* [11]. It is equivalent to minimizing the average evacuation time, and is motivated by the desire to minimize the transportation cost of evacuation and the total amount of psychological duress suffered by the evacuees, etc. It is more difficult than the completion time variety because the objective cost function is not unimodal along the given path. The minimization of the evacuation completion time (resp. aggregate evacuation time) reduces to the center (resp. median) problem, when the edge capacities are infinite, but finite capacities can cause *congestion* [5] which complicates the problems. To the best of our knowledge very little is known about this problem, except [2, 11, 13]. It is recently shown by Benkoczi et al. [2] that an aggregate time k-sink in path networks can be found in $O(kn \log^3 n)$ (resp. $O(kn^2 \log^2 n)$) time, if edge capacities are uniform (resp. nonuniform).

The main contribution of this paper is to find an aggregate time 1-sink that minimizes regret in $O(n^2 \log^2 n)$ time, improving the required time from $O(n^3)$ in [11]. A set of $O(n^2)$ *dominating* scenarios was identified in [11]. We first compute the aggregate time sinks for these scenarios, then the upper envelope of the "regret functions" of all these scenarios. Finally, we compute the lowest point of the upper envelope, which corresponds to the optimal sink μ^*. We make use of a few novel ideas. One is used in Sec. 4 to compute an aggregate time sink under each of the $O(n^2)$ *pseudo-bipartite* scenarios [11] in amortized $O(\log^2 n)$ time per sink. Another is used in Sec. 5 to compute the upper envelope of $O(n^2)$ regret functions (with $O(n^3)$ linear segments in total) in $O(n^2 \log^2 n)$ time, taking advantage of a special relationship among the regret functions.

In the next section, we define the terms that are used throughout this paper, and review some known facts which are relevant to later discussions. Sec. 3 discusses preprocessing which makes later operations more efficient. In Sec. 4 we show how to compute an aggregate time sink under scenarios that "dominate" others. We then compute in Sec. 5 an optimum sink that minimizes the max regret. The proofs of some lemmas could not be included due to space limitation. The interested reader is referred to the arXived version [3], which provides the proofs of all the lemmas and formal statements of three algorithms.

2 Preliminaries

2.1 Notations/definitions

Let $P(V, E)$ denote a given path network with vertex set $V = \{v_1, v_2, \ldots, v_n\}$. We assume that the vertices are arranged from left to right horizontally in the index order. For $1 \le i \le n-1$, there is an edge $e_i = (v_i, v_{i+1}) \in E$, whose length is denoted by $d(e_i)$. We write $p \in P$ for any point p (on an edge or vertex) of P, and for two points $a, b \in P$, we write $a \prec b$ or $b \succ a$ if a lies to the left of b. The distance between them is denoted by $d(a, b)$. If a and/or b lies on an edge, the distance is prorated. The capacity (the upper limit on the flow rate) of each edge is c (a constant), and the transit time is τ per unit distance. For $1 \le i \le j \le n$, $P[v_i, v_j]$ denotes the subpath of P from v_i to v_j.

For vertex v_i, $w(v_i) \in \mathbb{R}_+$ (the set of the positive reals) denotes its *weight*, which represents the number of "evacuees" initially located at v_i. Under *scenario* s, vertex v_i has a weight $w^s(v_i)$ such that $\underline{w}(v_i) \le w^s(v_i) \le \overline{w}(v_i)$, where $\underline{w}(v_i)$ and $\overline{w}(v_i)$ are assumed to be known. We define the Cartesian product $\mathcal{S} \triangleq \prod_{i=1}^{n} [\underline{w}(v_i), \overline{w}(v_i)]$, and consider each member of $\mathcal{S}$ as a scenario. Most of the above definitions were introduced in [5].

Our objective function under scenario s, $\Phi^s(x)$, is the sum of the evacuation times (sometimes called *cost*) of all the individual evacuees to point x. More formally, for $v_i \prec x \preceq v_{i+1}$ (resp. $v_i \preceq x \prec v_{i+1}$), let $\Phi^s_L(x)$ (resp. $\Phi^s_R(x)$) denote the cost at x for the evacuees from the vertices on $P[v_1, v_i]$ (resp. $P[v_{i+1}, v_n]$). We thus have $\Phi^s(x) \triangleq \Phi^s_L(x) + \Phi^s_R(x)$. Let $\mu^s \triangleq \mathrm{argmin}_x \Phi^s(x)$ be an aggregate time sink under s. Then $R^s(x) \triangleq \Phi^s(x) - \Phi^s(\mu^s)$ is called *regret* at x under s [15]. We say that scenario s' *dominates* scenario s *at* point x if $R^{s'}(x) \ge R^s(x)$ holds. The max regret at x is given by $R_{max}(x) \triangleq \max_{s \in \mathcal{S}} R^s(x)$ [15]. Our goal is to find a 1-sink, $x = \mu^*$, that minimizes $R_{max}(x)$.

By $\overline{s}_i$ we denote the scenario under which $w(v_j) = \overline{w}(v_j)$ for all $j \le i$ and $w(v_j) = \underline{w}(v_j)$ for all $j > i$, where $0 \le i \le n$. Similarly, by $\underline{s}_i$ we denote the scenario under which $w(v_j) = \underline{w}(v_j)$ for all $j \le i$ and $w(v_j) = \overline{w}(v_j)$ for all $j > i$. We call $\overline{s}_i$ and $\underline{s}_i$ *bipartite* scenarios. Finally, we define weight arrays $\underline{W}[v_i] \triangleq \sum_{k=1}^{i} \underline{w}(v_k)$ and $\overline{W[v_i]} \triangleq \sum_{k=1}^{i} \overline{w}(v_k)$, which can be precomputed in $O(n)$ time for all i, $1 \le i \le n$.

2.2 Clusters

In order to analyze congestion, in this subsection we review the notion of a *cluster* [11], and introduce some new related concepts, which play important roles in subsequent discussions. Given a point $x \in P$, which is not the sink, the evacuee flow at x toward the sink is a function of time, in general, alternating between no flow and flow at the rate limited by capacity c. A maximal group of vertices that provide uninterrupted flow without any gap forms a cluster. Such a cluster observed on edge $e_{k-1} = (v_{k-1}, v_k)$, arriving from right via v_k, is called an $\mathcal{R}^s$*-cluster with respect to* (any point on) e_{k-1}, including v_{k-1}, but excluding v_k. The vertex of such a cluster that is closest to e_{k-1} is called its *head vertex*. An $\mathcal{L}^s$*-cluster with respect to* e_k, including v_{k+1}, is similarly defined for evacuees arriving from left toward the sink.

If a cluster C contains a vertex v, the cluster is said to *carry* the evacuees from v. We now define particular clusters and cluster sequences.

- $C^s_{R,k}(v_i) \triangleq \mathcal{R}^s$-cluster with respect to e_{k-1} that contains vertex v_i $(i \ge k)$.
- $\mathcal{C}^s_{R,k}$: sequence of all $\mathcal{R}^s$-clusters with respect to e_{k-1} $(k = 2, \ldots, n)$.
- $C^s_{L,k}(v_i) \triangleq \mathcal{L}^s$-cluster with respect to e_k that contains vertex v_i $(i \le k)$.
- $\mathcal{C}^s_{L,k}$: sequence of all $\mathcal{L}^s$-clusters with respect to e_k $(k = 1, \ldots, n-1)$.

The total weight under scenario s of the vertices contained in cluster C is denoted by $\lambda^s(C)$. From now on we mainly discuss $\mathcal{R}^s$-clusters, since $\mathcal{L}^s$-clusters have analogous, symmetric properties. According to the above definition, $C^s_{R,k}(v_k)$ is the first cluster of sequence $\mathcal{C}^s_{R,k}$. If v_h and v_i ($v_h \prec v_i$) are the head vertices of two adjacent clusters in $\mathcal{C}^s_{R,k}$, then the following holds.

$$d(v_h, v_i)\tau > \lambda^s(C^s_{R,k}(v_h))/c. \tag{1}$$

Intuitively, this means that when the first evacuee from v_i arrives at v_h, all evacuees carried by $C^s_{R,k}(v_h)$ have left v_h already. For $v_{k-1} \preceq x \prec v_k$, let us analyze the cost of $C^s_{R,k}(v_i)$ at x, where $v_i \succeq v_k$. For the $\lambda^s(C^s_{R,k}(v_i))$ evacuees to move to x, let us divide the time required into two parts. The first part, called the *intra cost* [2], is the weighted waiting time before departure from the head vertex, v_j, of $C^s_{R,k}(v_i)$, and can be expressed as

$$\{\lambda^s(C^s_{R,k}(v_i))\}^2/2c. \tag{2}$$

Intuitively, (2) can be interpreted as follows. As far as the travel time to v_j and the waiting time at v_j are concerned, we may assume that all the $\lambda^s(C^s_{R,k}(v_i))$ evacuees were at v_j to start with. Since evacuees leave v_j at the rate of c, the mean wait time for the evacuees carried by $C^s_{R,k}(v_i)$ is $\lambda^s(C^s_{R,k}(v_i))/2c$, and thus the total for all of them is $\lambda^s(C^s_{R,k}(v_i))/2c \times \lambda^s(C^s_{R,k}(v_i)) = \{\lambda^s(C^s_{R,k}(v_i))\}^2/2c$. Note that the intra cost does not depend on x, as long as $v_{k-1} \preceq x \prec v_k$. This formula is accurate only when it is an integer, but for simplicity, we *adopt* (2) as our intra cost [5].[1]

The second part, called the *extra cost* [2], is the total transit time from the head vertex v_j of $C^s_{R,k}(v_i)$ to x for all the evacuees carried by $C^s_{R,k}(v_i)$, and can be expressed as

$$d(x, v_j)\lambda^s(C^s_{R,k}(v_i))\tau. \tag{3}$$

For the evacuees carried by $C^s_{L,k}(v_i)$, moving to the right, we similarly define its intra cost and extra cost, where $v_i \preceq v_k \prec x \preceq v_{k+1}$. For $v_{k-1} \preceq x \prec v_k$, we now introduce a cost function for cluster sequence $\mathcal{C}^s_{R,k}$.

$$\Phi^s_{R,k}(x) \quad \triangleq \quad \sum_{C \in \mathcal{C}^s_{R,k}} d(x, v_i)\lambda^s(C)\tau + \sum_{C \in \mathcal{C}^s_{R,k}} \lambda^s(C)^2/2c. \tag{4}$$

We name the first (resp. second) term in (4) $E^s_{R,k}$ (resp. $I^s_{R,k}$). Similarly, for x ($v_k \prec x \preceq v_{k+1}$), we define

$$\Phi^s_{L,k}(x) \quad \triangleq \quad \sum_{C \in \mathcal{C}^s_{L,k}} d(v_i, x)\lambda^s(C)\tau + \sum_{C \in \mathcal{C}^s_{L,k}} \lambda^s(C)^2/2c \triangleq E^s_{L,k} + I^s_{L,k}. \tag{5}$$

When v_k is clear from the context, or when there is no need to refer to it, we may write $\Phi^s_R(x)$ (resp. $\Phi^s_L(x)$) to mean $\Phi^s_{R,k}(x)$ (resp. $\Phi^s_{L,k}(x)$). The aggregate of the evacuation times to x of all evacuees is given by

$$\Phi^s(x) = \begin{cases} \Phi^s_{L,k}(x) + \Phi^s_{R,k+1}(x) & \text{for } v_k \prec x \prec v_{k+1} \\ \Phi^s_{L,k-1}(x) + \Phi^s_{R,k+1}(x) & \text{for } x = v_k. \end{cases} \tag{6}$$

A point x that minimizes $\Phi^s(x)$ is called an *aggregate time sink*, a.k.a. *minsum sink*, under s. An aggregate time sink shares the following property of a *median* [14].

▶ **Lemma 1** ([13]). *Under any scenario, there is an aggregate time sink at a vertex.*

[1] It is accurate for fluid-like "evacuees" that is always divisible by capacity c.

2.3 What is known

▶ **Lemma 2** ([11]). *For any given scenario $s \in \mathcal{S}$,*
(a) *We can compute $\{\Phi_L^s(v_i), \Phi_R^s(v_i) \mid i = 1, \ldots, n\}$ in $O(n)$ time.*
(b) *We can compute μ^s and $\Phi^s(\mu^s)$ in $O(n)$ time.*

A scenario s under which all vertices on the left (resp. right) of a vertex have the max (resp. min) weights is called an *L-pseudo-bipartite* scenario [11]. The vertex v_b, where $1 \le b \le n$, that may take an intermediate weight $\underline{w}(v_b) \le w(v_b) \le \overline{w}(v_b)$, is called the *boundary vertex*, a.k.a. *intermediate vertex* [11]. Let $b(s)$ denote the index of the boundary vertex under pseudo-bipartite scenario s. We consider the bipartite scenarios, under which $w(v_b) = \underline{w}(v_b)$ and $w(v_b) = \overline{w}(v_b)$, also as special pseudo-bipartite scenarios, and in the former (resp. latter) case, either $b(s) = b - 1$ or $b(s) = b$ (resp. $b(s) = b$ or $b(s) = b + 1$). The vertices that have the maximum (resp. minmum) weights comprise the *max-weighted* (resp. *min-weighted*) part. We define an *R-pseudo-bipartite* scenario symmetrically with the max-weighted part and the min-weighted part reversed. As $w(v_b)$ increases from $\underline{w}(v_b)$ to $\overline{w}(v_b)$, clusters may merge.

Weight $w^s(v_b)$ is called a *critical weight*, if two clusters with respect to *any point merge* as $w(v_b)$ increases to become a scenario s. Let $\mathcal{S}_L^*$ (resp. $\mathcal{S}_R^*$) denote the set of the L- (resp. R-)pseudo-bipartite scenarios that correspond to the critical weights. Thus each scenario in $\mathcal{S}_L^*$ (resp. $\mathcal{S}_R^*$) can be specified by v_b and $w(v_b)$. Let $\mathcal{S}^* \triangleq \mathcal{S}_L^* \cup \mathcal{S}_R^*$.

▶ **Lemma 3** ([11]).
(a) *Any scenario in $\mathcal{S}$ is dominated at every point x by a scenario in $\mathcal{S}^*$.*[2]
(b) *$|\mathcal{S}^*| = O(n^2)$, and all scenarios in $\mathcal{S}^*$ can be determined in $O(n^2)$ time.*

2.4 Road map

From now on, we proceed as follows.
(1) Investigate important properties of clusters to prepare for later sections. (Sec. 3)
(2) Compute $\{\mu^s \mid s \in \mathcal{S}^*\}$ in $O(n^2 \log^2 n)$ time. (Sec. 4)
(3) Compute $R_{max}(x) = \max\{R^s(x) \mid s \in \mathcal{S}^*\}$ in $O(n^2 \log^2 n)$ time. (Sec. 5.1) $R_{max}(x)$ is a piecewise linear function, and can be specified by the set of its bending points.
(4) Find point $x = \mu^*$ that minimizes $R_{max}(x)$ in $O(n^2)$ time. (Sec. 5.2)

3 Clusters under pseudo-bipartite scenarios

3.1 Preprocessing

Without loss of generality, we concentrate on $\mathcal{R}^s$-clusters for $s \in \mathcal{S}_L^*$, since the other combinations, such as $\mathcal{R}^s$-clusters for $s \in \mathcal{S}_R^*$, etc., can be treated analogously. For $k = 2, \ldots, n$, let $\mathcal{C}_{R,k}^s$ consist of $q^s(k)$ clusters

$$\mathcal{C}_{R,k}^s = \langle C_1, C_2, \ldots, C_{q^s(k)} \rangle, \tag{7}$$

and let u_i be the head vertex of C_i, where $v_k = u_1 \prec \ldots \prec u_{q^s(k)}$. By (1), $d(u_i, u_{i+1})\tau > \lambda(C_i)/c$ holds for $i = 1, 2, \ldots, q^s(k) - 1$.

[2] Not necessarily by the same scenario. The scenario depends on a particular x.

▶ **Lemma 4.**

(a) *For any scenario $s \in \mathcal{S}$, the number of distinct clusters in $\{\mathcal{C}^s_{R,k} \mid k = 2, \dots, n\}$ is $O(n)$.*

(b) *For any scenario $s \in \mathcal{S}$, we can construct $\{\mathcal{C}^s_{R,k} \mid k = 2, \dots, n\}$ in $O(n)$ time.*

Proof. (a) Consider $\mathcal{C}^s_{R,k}$ in the order $k = n, n-1 \dots, 2$. Cluster sequence $\mathcal{C}^s_{R,v_n}$ consists of just one cluster composed of v_n. Let $\mathcal{C}^s_{R,k+1} = \langle C'_1, C'_2 \dots, C'_{q^s(k+1)} \rangle$ for some k. Cluster $C_1 \in \mathcal{C}^s_{R,k}$ contains vertex v_k and possibly the vertices of $C'_1, \dots, C'_h$ for some h, where $0 \leq h \leq q^s(k+1)$ and $h = 0$ means C_1 contains just v_k and no other vertex. Note that C_1 is new when we go from $k+1$ to k, but the other clusters of $\mathcal{C}^s_{R,k}$, i.e., $C_2, \dots, C_{q^s(k)}$ are $C'_{h+1}, \dots, C'_{q^s(k+1)}$. This means that each k introduces just one new cluster, and thus the number of distinct clusters is $O(n)$.

(b) Let us construct $\mathcal{C}^s_{R,k}$ in the order $k = n, n-1 \dots, 2$. Assume that we have computed $\mathcal{C}^s_{R,k+1}$, and want to compute C_1. If (1) does not hold between the new singleton cluster v_k and the first cluster, C'_1, of $\mathcal{C}^s_{R,k+1}$, namely if $d(v_k, v_{k+1})\tau \leq \lambda^s(\{v_k\})/c$, then v_k and C'_1 merge to form a single cluster. (1) may become violated for this new cluster and C'_2, in which case they also merge. As a result of such chain reaction, if v_k merges with the first h clusters in $\mathcal{C}^s_{R,k+1}$ this way, we spend $O(h)$ time in computing C_1. Those h clusters will never contribute to the computation time from now on. If we pay attention to the head vertex, u_i, of C_i, it gets absorbed into a larger cluster at most once, and each time such an event takes place, constant computation time incurs. ◀

Computing the extra cost $E^s_{R,k}$ in (4) is fairly easy, because the extra cost of cluster C is linear in $\lambda^s(C)$. The intra costs can also be computed efficiently.

▶ **Lemma 5** ([11])**.** *Given a scenario $s \in \mathcal{S}$,*

(a) *We can compute $\{E^s_{R,k}, I^s_{R,k} \mid k = 1, \dots, n-1\}$ in $O(n)$ time.*

(b) *We can compute $\{E^s_{L,k}, I^s_{L,k} \mid k = 2, \dots, n\}$ in $O(n)$ time.*

Let $s_0 \triangleq \underline{s}_n$ and $s_M \triangleq \overline{s}_n$. The following corollary follows easily from Lemmas 4 and 5.

▶ **Corollary 6.**

(a) *There are $O(n)$ distinct clusters among the cluster sequences in $\{\mathcal{C}^{s_0}_{L,k} \cup \mathcal{C}^{s_0}_{R,k} \cup \mathcal{C}^{s_M}_{L,k} \cup \mathcal{C}^{s_M}_{R,k} \mid k = 1, \dots, n\}$, and we can compute them in $O(n)$ time.*

(b) *We can compute $\{E^{s_0}_{R,k}, I^{s_0}_{R,k}, E^{s_M}_{R,k}, I^{s_M}_{R,k} \mid k = 1, \dots, n-1\}$ and $\{E^{s_0}_{L,k}, I^{s_0}_{L,k}, E^{s_M}_{L,k}, I^{s_M}_{L,k} \mid k = 1, \dots, n-1\}$ in $O(n)$ time*

(c) *For each cluster sequence in $\mathcal{C}^{s_0}_{L,k} \cup \mathcal{C}^{s_0}_{R,k} \cup \mathcal{C}^{s_M}_{L,k} \cup \mathcal{C}^{s_M}_{R,k}$, we can compute the prefix sum of the intra costs in $O(n)$ time. Thus we can compute the prefix sums of the intra costs for all k in $O(n^2)$ time, if we do not repeat the common data.*

From now on, we assume that we have precomputed all the data mentioned in Corollary 6.

3.2 Constructing set of pseudo-bipartite scenarios $\mathcal{S}^*$

Let $s = s_0$ in (7). Starting with $b = k$, we increase $w(v_b)$ until $C^{s_0}_{R,k}(v_k)$ merges with the next cluster in $\mathcal{C}^{s_0}_{R,k}$, and record the value of b and the amount of increase δ above $\underline{w}(v_b)$ that caused this merger. We repeat this with the newly formed cluster, instead of $C^{s_0}_{R,k}(v_k)$. If $\overline{w}(v_b)$ is reached we fix $w(v_b) = \overline{w}(v_b)$, increment b and repeat, as long as $v_b \in C^{s_M}_{R,k}(v_k)$ holds. We will end up with a list

$$\Delta_{R,k} \triangleq \{(b_1, \delta_{k,1}), (b_2, \delta_{k,2}), \dots\}, \tag{8}$$

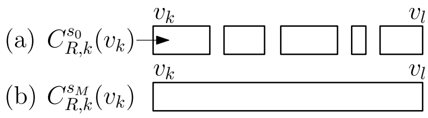

Figure 1 (a) Some clusters in $\mathcal{C}^{s_0}_{R,k}$; (b) $C^{s_M}_{R,k}(v_k)$.

where $k \le b_1 \le b_2 \le \cdots$, and for any two adjacent items, $(b_i, \delta_{k,i})$ and $(b_{i+1}, \delta_{k,i+1})$, if $b_i = b_{i+1}$ then $\delta_{k,i} < \delta_{k,i+1}$. Intuitively, $(b_i, \delta_{k,i}) \in \Delta_{R,k}$ means that when $w(v_{b_i}) = \underline{w}(v_{b_i}) + \delta_{k,i}$ the first cluster of $C^s_{R,k}(v_k)$ expands by merging with the next cluster, where s is the scenario reflecting the weight changes made so far. Fig. 1(a) illustrates some clusters in the beginning of $\mathcal{C}^{s_0}_{R,k}$, and Fig. 1(b) shows $C^{s_M}_{R,k}(v_k)$. We start with $v_b = v_k$ in $C^{s_0}_{R,k}(v_k)$ in Fig. 1(a), which is a part of $\mathcal{C}^{s_0}_{R,k}$ that we already have. We increase $w(v_b)$ by $\delta_{k,1}$ from $w^{s_0}(v_b) = \underline{w}(v_b)$ until $C^{s_0}_{R,k}(v_k)$ expands by merging with the next cluster on its right. This $\delta_{k,1}$ is obtained by solving[3]

$$d(u_1, u_2)\tau = \{\lambda^{s_0}(C^{s_0}_{R,k}(v_k)) + \delta_{k,1}\}/c. \tag{9}$$

Assuming $\underline{w}(v_b) + \delta_{k,1} \le \overline{w}(v_b)$, for $w^s(v_b) = \underline{w}(v_b) + \delta_{k,1}$, $C^{s_0}_{R,k}(v_k)$ may merge with the next $h-1$ clusters in $\mathcal{C}^{s_0}_{R,k}$, where $h \ge 2$, resulting in a combined cluster C under s ($\neq s_0$), and the first item $(k, \delta_{k,1})$ being created in $\Delta_{R,k}$. If $\underline{w}(v_b) + \delta_{k,1} \le \overline{w}(v_b)$, on the other hand, we repeat this operation to find the increment $\delta_{k,2}$, if any, above $\underline{w}(v_b)$ that causes C to absorb the $h + 1^{st}$ cluster in $\mathcal{C}^{s_0}_{R,k}$, etc. If $\underline{w}(v_b) + \delta_{k,1} > \overline{w}(v_b)$, on the other hand, we set $w(v_b) = \overline{w}(v_b)$ and increment b by one without recording $\delta_{k,1}$. When this process terminates, we end up with $C^{s_M}_{R,k}(v_k)$ in Fig. 1(b), because all the vertices involved now have their max weights, and we will have constructed $\Delta_{R,k}$.[4] Clearly, each item $(b_j, \delta_{k,j}) \in \Delta_{R,k}$ corresponds to a scenario $s_j \in \mathcal{S}^*_L$ in the following way.

$$w^{s_j}(v_i) = \begin{cases} w^{s_M}(v_i) & \text{for } 1 \le i < b_j \\ \underline{w}(v_{b_j}) + \delta_{k,j} & \text{for } i = k \\ w^{s_0}(v_i) & \text{for } b_j < i \le n \end{cases} \tag{10}$$

Let $\mathcal{S}^*_{L,k}$ denote the set of scenarios corresponding to the increments in $\Delta_{R,k}$ according to (6). It is clear that $\mathcal{S}^*_L = \cup_{k=1}^n \mathcal{S}^*_{L,k}$. Note that under any $s \in \mathcal{S}^*_{L,k}$, $C^s_{R,k}(v_{b(s)})$ is the first cluster in $\mathcal{C}^s_{R,k}$.

▶ **Lemma 7.**

(a) *We can compute $\Delta_{R,k}$ in $O(|C^{s_M}_{R,k}(v_k)|)$ time, where $|C^{s_M}_{R,k}(v_k)|$ denotes the number of vertices in cluster $C^{s_M}_{R,k}(v_k)$.*

(b) *We can construct $\{\Delta_{R,k} \mid k = 2, \ldots, n\}$, hence $\mathcal{S}^*_L$, in $O(n^2)$ time.*

(c) *For each scenario $s \in \mathcal{S}^*_{L,k}$, we can identify the last vertex in $C^s_{R,k}(v_k)$ in constant extra time while computing $\Delta_{R,k}$.*

[3] Let u_i be as defined after (7) for $s = s_0$.
[4] The above method to compute $\Delta_{R,k}$ is presented as a formal algorithm in [3].

4 Computing sinks $\{\mu^s \mid s \in \mathcal{S}^*\}$

4.1 Computing $\{\Phi^s(x) \mid s \in \mathcal{S}^*\}$

Let us now turn our attention to the computation of the extra and intra costs under the scenarios in $\mathcal{S}^*_{L,k}$. Those under the scenarios in $\mathcal{S}^*_{R,k}$ can be computed similarly. While computing $\Delta_{R,k}$ as in Sec. 3.2, we can update the extra and intra costs at v_k under the corresponding scenario $s \in \mathcal{S}^*_{L,k}$ as follows.

When the first increment $\delta_{k,1}$ causes the merger of the first two clusters in $\mathcal{C}^{s_0}_{R,k}$, for example, we subtract the extra cost contributions of those two clusters from $E^{s_0}_{R,k}$, and add the new contribution from the merged cluster in order to compute $E^s_{R,k}$ for the new scenario s that results from the incremented weight $\underline{w}^s(v_k) = \underline{w}(v_k) + \delta_{k,1}$. We can similarly compute $I^s_{R,k}$ from $I^{s_0}_{R,k}$ in constant time. Carrying out these operations whenever a newly expanded cluster is created thus takes $O(n)$ time for a given k and $O(n^2)$ time in total for all k's. Define $\Delta_R \triangleq \cup_{k=2}^n \Delta_{R,k}$.

▶ **Lemma 8.** *Assume that Δ_R, as well as all the data mentioned in Corollary 6, are available. Then under any given scenario $s \in \mathcal{S}^*_L$, we can compute the following in $O(\log n)$ time.*
(a) *$\Phi^s(v_i) = \Phi^s_L(v_i) + \Phi^s_R(v_i)$ for any given index i.*
(b) *$\Phi^s(x) = \Phi^s_L(x) + \Phi^s_R(x)$ for any given point x.*

Among the items in Δ_R, there is a natural lexicographical order, ordered first by b and then by $w(v_b)$, from the smallest to the largest. We write $s \lessdot s'$ if s is ordered before s' in this order. In what follows we assume the items in Δ_R are sorted by $\lessdot$.

4.2 Tracking sink μ^s

Observe that we have $\Phi^s_L(x) = \Phi^{s_M}_L(x)$ for $x \preceq v_b$, which is independent of $w(v_b)$. Similarly, we have $\Phi^s_R(x) = \Phi^{s_0}_R(x)$ for $x \succeq v_b$, which is also independent of $w(v_b)$. We initialize the current scenario by $s = s_0$, the boundary vertex v_b by $b = 1$, and its weight by $w(v_b) = w^{s_0}(v_1)$. For each successive increment in Δ_R, from the smallest (according to $\lessdot$), we want to know the leftmost 1-sink under the corresponding scenario. It is possible that, as we increase the weight $w(v_b)$, the sink may jump across v_b from its right side to its left side, and vice versa, back and forth many times. We shall see how this can happen below.

By Lemma 8, for a given index b, we can compute $\{\Phi^{\overline{s}_{b-1}}(v_i) \mid i = 1, 2, \ldots, n\}$ in $O(n \log n)$ time.[5] We first scan $\Phi^{\overline{s}_{b-1}}(v_b), \Phi^{\overline{s}_{b-1}}(v_{b-1}), \ldots, \Phi^{\overline{s}_{b-1}}(v_1)$, and whenever we encounter a value smaller than those we examined so far, we record the index of the corresponding vertex. Let $\mathcal{I}^b_L$ be the recorded index set, starting with b. We then scan $\Phi^{\overline{s}_{b-1}}(v_b), \Phi^{\overline{s}_{b-1}}(v_{b+1}), \ldots,$ $\Phi^{\overline{s}_{b-1}}(v_n)$ similarly, and let $\mathcal{I}^b_R$ be the recorded index set, starting with b. We now plot point $(v_i, \Phi^{\overline{s}_{b-1}}(v_i))$ for $i \in \mathcal{I}^b_L \cup \mathcal{I}^b_R$ in the x-y coordinate system, with distance $d(v_1, v_i)$ as the x value and $\Phi^{\overline{s}_{b-1}}(v_i)$ as the y value. See Fig. 2, where $d(v_1, v_i)$ is indicated by v_i. It is clear from the definition that for $i, j \in \mathcal{I}^b_L$ such that $i < j$, we have $\Phi^{\overline{s}_{b-1}}(v_i) < \Phi^{\overline{s}_{b-1}}(v_j)$, and for $i, j \in \mathcal{I}^b_R$ such that $i < j$, we have $\Phi^{\overline{s}_{b-1}}(v_i) > \Phi^{\overline{s}_{b-1}}(v_j)$. Therefore, the points plotted on the left (resp. right) side of v_b get higher and higher as we approach v_b from left (resp. right), as seen by the black dots in Fig. 2.

Note that for a vertex v_i ($\prec v_b$), as $w(v_b)$ is increased, $\Phi^s_R(v_i)$ increases, while $\Phi^s_L(v_i)$ remains fixed at $\Phi^{s_M}_L(v_i)$, where s is the scenario reflecting the change in $w(v_b)$. For $v_i \succ v_b$, on the other hand, as $w(v_b)$ is increased, $\Phi^s_L(v_i)$ increases, while $\Phi^s_R(v_i)$ remains fixed at

[5] Recall the definition of $\overline{s}_j$ from Sec. 2.1.

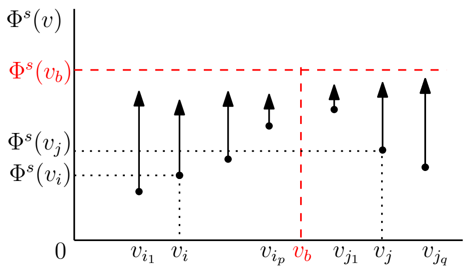

Figure 2 Graphical representation of $\Phi^{\overline{s}_{b-1}}(v_i) = \Phi_L^{\overline{s}_{b-1}}(v_i) + \Phi_R^{\overline{s}_{b-1}}(v_i)$.

$\Phi_R^{s_0}(v_i)$. A vertical arrow in Fig. 2 indicates the relative amount of increase in the cost at the corresponding vertex when $w(v_b)$ is increased by a certain amount. Note that the farther away a vertex is from v_b, the more is the increase in the cost.

The following proposition summarizes the above observations.

▶ **Proposition 9.**
(a) $\Phi^s(v_i) < \Phi^s(v_j)$ *holds for any pair* $i, j \in \mathcal{I}_L^b$ *such that* $i < j$.
(b) $\Phi^s(v_i) > \Phi^s(v_j)$ *holds for any pair* $i, j \in \mathcal{I}_R^b$ *such that* $i < j$.
(c) *Either the vertex with the smallest index in* $\mathcal{I}_L^b$ *or the vertex with the largest index in* $\mathcal{I}_R^b$ *has the lowest cost, i.e., it is a 1-sink.*

Note that the cost at v_b, $\Phi_R^s(v_b)$, is the highest among the points plotted, and is not affected by the change in $w(v_b)$. We consider the three properties in Proposition 9 as *invariant* properties, and remove the vertices that do not satisfy (a) or (b), as we increase $w(v_b)$. As we increase $w(v_b)$, in the order of the sorted increments in Δ_R, we update $\mathcal{I}_L^b$ and $\mathcal{I}_R^b$, looking for the change of the sink.

▶ **Proposition 10.** *As* $w(v_b)$ *is increased, there is a sink at the same vertex for all the increments tested since the last time the sink moved, until the smallest index in* $\mathcal{I}_L^b$ *or the largest index in* $\mathcal{I}_R^b$ *changes, causing the sink to move again. The sink cannot move away from* v_b.

We are thus interested in how $\mathcal{I}_L^b$ and $\mathcal{I}_R^b$ change, in particular, when its smallest index in $\mathcal{I}_L^b$ or the largest index in $\mathcal{I}_R^b$ changes.

▶ **Lemma 11.** *Let* i *and* j *be vertex indices such that either they are adjacent in* $\mathcal{I}_L^b$ *and* $i < j$ *holds, or adjacent in* $\mathcal{I}_R^b$ *and* $i > j$ *holds. The smallest*[6] $(b, \delta) \in \Delta_R$, *if any, such that increasing* $w(v_b)$ *by* δ *above* $\underline{w}(v_b)$ *causes the cost at* v_i *to reach or exceed that at* v_j *can be determined in* $O(\log^2 n)$ *time.*

Proof. Use binary search on Δ_R (sorted by $\lessdot$), and compare the costs at v_i and v_j for each probe in $O(\log n)$ time, using Lemma 8. ◀

If such a δ in Lemma 11 does not exist, we set $\delta = \infty$. From Lemma 11, it follows that the total time for all adjacent pairs is $O(n \log^2 n)$. We insert a triple $(\delta; i, j)$ into a *min-heap* $\mathcal{H}_b$, organized according to the first component δ, from which we can extract the item with the smallest first component. For a given b, once $\mathcal{H}_b$ has been constructed this way, we pop the item $(\delta; i, j)$ with the smallest δ from $\mathcal{H}_b$ in constant time. If $i, j \in \mathcal{I}_L^b$ (resp. $i, j \in \mathcal{I}_R^b$) then

[6] According to $\lessdot$.

we remove i (resp. j) from $\mathcal{I}_L^b$ (resp. $\mathcal{I}_R^b$), and compute $(\delta'; i^-, j)$ (resp. $(\delta'; i, j^+)$) where i^- (resp. j^+) is the index in $\mathcal{I}_L^b$ (resp. $\mathcal{I}_R^b$) that is immediately before (resp. after) i (resp. j). By Lemma 11 we can find δ' in $O(\log^2 n)$ time, and insert $(\delta'; i^-, j)$ (resp. $(\delta'; i, j^+)$) into $\mathcal{H}_b$, taking $O(\log n)$ time. If i was the smallest index in $\mathcal{I}_L^b$, the sink may have moved. In this case no new item is inserted into $\mathcal{H}_b$. Similarly, if j was the largest index in $\mathcal{I}_R^b$, the sink may have moved, and no new item is inserted into $\mathcal{H}_b$.

We repeat this until either $\mathcal{H}_b$ becomes empty or the minimum δ-value in $\mathcal{H}_b$ is ∞. It is repeated $O(n)$ times, so the total time required is $O(n \log^2 n)$. If the sink moves when the smallest index in $\mathcal{I}_L^b$ or the largest index in $\mathcal{I}_R^b$ changes, we have determined the sink under all the scenarios with the lighter $w(v_b)$ since the last time the sink moved. Once $w(v_b) = \underline{w}(v_b) + \delta$ reaches $\overline{w}(v_b)$, b is incremented, and the new boundary vertex now lies to the right of the old boundary vertex v_b in Fig. 2. For each $b = 1, 2, \ldots, n$, let $\mathcal{S}_b = \{s \in \mathcal{S}^* \mid b(s) = b\}$.[7]

▶ **Lemma 12.**
(a) *Sinks* $\{\mu^s \mid s \in \mathcal{S}_b \cap \mathcal{S}_L^*\}$ *can be computed in* $O(n \log^2 n)$ *time for a given boundary vertex* v_b.
(b) *Sinks* $\{\mu^s \mid s \in \mathcal{S}_b \cap \mathcal{S}_R^*\}$ *can be computed in* $O(n \log^2 n)$ *time for a given boundary vertex* v_b.

For the clusters in $\mathcal{C}_{R,i}^s$ that lie to the right of $C_{R,i}^s(v_b)$ and are not merged as a result of an increase in $w(v_b)$, the sum of their intra costs was already precomputed. Repeating the above operations for $b = 1, 2, \ldots, n$, we get our first major result.

▶ **Lemma 13.** *The sinks* $\{\mu^s \mid s \in \mathcal{S}^*\}$ *can be computed in* $O(n^2 \log^2 n)$ *time.*

5 Minmax regret sink

Now that we know how to compute the sinks $\{\mu^s \mid s \in \mathcal{S}^*\}$, we proceed to compute the upper envelope for the $O(n^2)$ regret functions $\{R^s(x) = \Phi^s(x) - \Phi^s(\mu^s) \mid s \in \mathcal{S}^*\}$. The minmax regret sink μ^* is at the lowest point of this upper envelope.

5.1 Upper envelope for $\{R^s(x) \mid s \in \mathcal{S}^*\}$

If we try to find the upper envelope $\max_{s \in \mathcal{S}^*} \Phi^s(x)$ in a naïve way, it would take at least $O(n^3)$ time, since $|\mathcal{S}^*| = O(n^2)$, and for each s, $\Phi^s(x)$ consists of $O(n)$ linear segments. We employ the following two-phase approach.
Phase 1: For each b, compute the upper envelope $\max_{s \in \mathcal{S}_b} R^s(x)$.
Phase 2: Compute the upper envelope for the results from *Phase 1*.
In *Phase 1*, we successively update the upper envelope, incorporating regret functions one at a time, which can be done in amortized $O(\log^2 n)$ time per regret function. Thus the total time for a given b is $O(n \log^2 n)$ and the total time for all b is $O(n^2 \log^2 n)$. In *Phase 2*, we then compute the upper envelope for the resulting $O(n)$ regret functions with a total of $O(n^2)$ linear segments in $O(n^2 \log n)$ time. To implement *Phase 1*, we first present the following lemma.

▶ **Lemma 14.** *Let* $s, s' \in \mathcal{S}_b$ *be two scenarios such that and* $s \lessdot s'$. *As* x *moves to the right, the difference* $D(x) = \Phi^{s'}(x) - \Phi^s(x)$ *decreases monotonically for* $v_1 \preceq x \preceq v_b$ *and increases monotonically for* $v_b \preceq x \preceq v_n$.

---

[7] The above method is presented as a formal algorithm in [3].

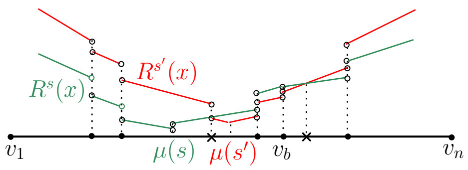

Figure 3 $R^s(x)$ and $R^{s'}(x)$ cross each other at two points in this example.

We divide each regret function in $\{R^s(x) \mid s \in \mathcal{S}_b\}$ into two parts: the left of v_b and the right of v_b. We then find the upper envelope for the left set and right set separately. Note that each $R^s(x)$ has $O(n)$ bending points, since they bend only at vertices. Taking the maximum of two such functions may add one extra bending point on an edge, so the total bending points in the upper bound is still $O(n)$. By definition we have

$$\begin{aligned} R^{s'}(x) - R^s(x) &= \Phi^{s'}(x) - \Phi^{s'}(\mu^{s'}) - \{\Phi^s(x) - \Phi^s(\mu^s)\} \\ &= \Phi^{s'}(x) - \Phi^s(x) - \{\Phi^{s'}(\mu^{s'}) - \Phi^s(\mu^s)\}. \end{aligned} \tag{11}$$

Since the second term in (11) is independent of position x, Lemma 14 implies

▶ **Lemma 15.** *Let $s, s' \in \mathcal{S}_b$ be two scenarios such that and $s \lessdot s'$. Then $R^{s'}(x)$ may cross $R^s(x)$ at most once in the interval $[v_1, v_b]$ from above, and at most once in the interval $[v_b, v_n]$ from below.*

See Fig. 3 for an illustration for Lemma 15. For $x \succ v_b$, we compute $\max_{s\in\mathcal{S}_b^*} R^s(x)$, updating a partially computed upper envelope $U(x)$ by successively incorporating the "next" regret function $R^s(x)$ to it. We can use binary search to find the crossing point of $U(x)$ and $R^s(x)$, and invoke Lemma 8.

▶ **Lemma 16.**

(a) *The upper envelope* $\max_{s\in\mathcal{S}_b} R^s(x)$ *has* $O(|\mathcal{S}_b| + n)$ *line segments.*

(b) *We can compute the upper envelope* $\max_{s\in\mathcal{S}_b} R^s(x)$ *in* $O(|\mathcal{S}_b| \log^2 n)$ *time.*

Proof. (a) Without loss of generality, consider the upper envelope in the interval $[v_b, v_n]$. Since $R^s(x) = \Phi^s(x) - \Phi^s(\mu^s)$, $R^s(x)$ is linear over the edge connecting any adjacent pair of vertices, and $\max_{s\in\mathcal{S}_b} \Phi^s(x)$ has $O(|\mathcal{S}_b| + n)$ line segments on all edges by Lemma 15.

(b) See the analysis of Algorithm 3 in [3]. ◀

5.2 Main theorem

Since $\cup_{b=1}^n \mathcal{S}_b = \mathcal{S}^*$ and $|\mathcal{S}^*| = O(n^2)$, Lemma 16 implies that it takes $O(n^2 \log^2 n)$ time to compute $\{\max_{s\in\mathcal{S}_b} R^s(x) \mid b = 1, \ldots, n\}$. These n upper envelope together have $O(n^2)$ linear segments. Hershberger [9] showed that the upper envelope of m line segments can be computed in $O(m \log m)$ time. We can use his method to compute the global upper envelope for $\{\max_{s\in\mathcal{S}_b} R^s(x) \mid b = 1, \ldots, n\}$ in $O(n^2 \log n)$ additional time.

▶ **Lemma 17.** *The upper envelope* $\max_{s\in\mathcal{S}^*} R^s(x)$ *can be computed in* $O(n^2 \log^2 n)$ *time.*

So far we have paid no attention to the negative spikes in $R^s(x)$ at vertices. Divide the problem in two subproblems: minmax regret sink is (i) on an edge, and (ii) at a vertex. Compare the two solutions and pick the one with the smaller cost. In addition to Lemma 17, we should evaluate the regret at each vertex. The true minmax regret sink is at the point with the minimum of these maximum regrets. Corollary 6 and Lemmas 3, 13 and 17 imply our main result.

▶ **Theorem 18.** *The minmax regret sink on path networks can be computed in $O(n^2 \log^2 n)$ time.*

6 Conclusion

We presented an $O(n^2 \log^2 n)$ time algorithm for finding a minmax regret aggregate time (a.k.a. minsum) sink on path networks with uniform edge capacities, which improves upon the previously most efficient $O(n^3)$ time algorithm in [11]. We hope some methods we devised in this paper will find applications in solving some other related problems. Future research topics include efficiently solving the minmax regret problem for aggregate time sink for more general networks such as trees. No such polynomial time algorithm is known at present.

References

1 Guru Prakash Arumugam, John Augustine, Mordecai Golin, and Prashanth Srikanthan. A Polynomial Time Algorithm for Minimax-Regret Evacuation on a Dynamic Path. *arXiv:1404,5448 v1 [cs.DS] 22 April*, 2014.

2 R. Benkoczi, B. Bhattacharya, Y. Higashikawa, T. Kameda, and N. Katoh. Minsum k-sink on dynamic flow path network. In *Combinatorial Algorithms (Iliopoulos, Costas, Leong, Hon Wai, Sung, Wing-Kin, Eds.), Springer-Verlag LNCS 110979*, pages 78–89, 2018.

3 Binay Bhattacharya, Yuya Higashikawa, Tiko Kameda, and Naoki Katoh. Minmax regret 1-sink for aggregate evacuation time on path networks. arXiv:1806.00814 Oct 2018.

4 Binay Bhattacharya and Tsunehiko Kameda. Improved algorithms for computing minmax regret sinks on path and tree networks. *Theoretical Computer Science*, 607:411–425, November 2015.

5 Siu-Wing Cheng, Yuya Higashikawa, Naoki Katoh, Guanqun Ni, Bing Su, and Yinfeng Xu. Minimax regret 1-sink location problem in dynamic path networks. In *Proc. Annual Conf. on Theory and Applications of Models of Computation (T-H.H. Chan, L.C. Lau, and L. Trevisan, Eds.), Springer-Verlag, LNCS 7876*, pages 121–132, 2013.

6 L.R. Ford and D.R. A. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

7 Mordecai Golin and Sai Sandeep. Minmax-regret k-sink location on a dynamic tree network with uniform capacities. *arXiv:1806.03814v1 [cs.DS] 11 June*, 2018.

8 H.W. Hamacher and S.A. Tjandra. Mathematical modelling of evacuation problems: a state of the art. *in: Pedestrian and Evacuation Dynamics, Springer Verlag,*, pages 227–266, 2002.

9 J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, 1989.

10 Yuya Higashikawa, John Augustine, Siu-Wing Cheng, Mordecai J. Golin, Naoki Katoh, Guanqun Ni, Bing Su, and Yinfeng Xu. Minimax regret 1-sink location problem in dynamic path networks. *Theoretical Computer Science*, 588(11):24–36, 2015.

11 Yuya Higashikawa, Siu-Wing Cheng, Tsunehiko Kameda, Naoki Katoh, and Shun Saburi. Minimax regret 1-median problem in dynamic path networks. *Theory of Computing Systems*, 62(6):1392–1408, August 2018.

12 Yuya Higashikawa, Mordecai J. Golin, and Naoki Katoh. Minimax regret sink location problem in dynamic tree networks with uniform capacity. *Journal of Graph Algorithms and Applications*, 18(4):539–555, 2014.

13 Yuya Higashikawa, Mordecai J. Golin, and Naoki Katoh. Multiple sink location problems in dynamic path networks. *Theoretical Computer Science*, 607(1):2–15, 2015.

14 Oded Kariv and S.Louis Hakimi. An algorithmic approach to network location problems, Part II: The p-median. *SIAM J. Appl. Math.*, 37:539–560, 1979.

15 P. Kouvelis and G. Yu. *Robust Discrete Optimization and its Applications.* Kluwer Academic Publishers, London, 1997.

16 Satoko Mamada, Takeaki Uno, Kazuhisa Makino, and Satoru Fujishige. An $O(n \log^2 n)$ algorithm for a sink location problem in dynamic tree networks. *Discrete Applied Mathematics*, 154:2387–2401, 2006.

17 Haitao Wang. Minmax Regret 1-Facility Location on Uncertain Path Networks. *European J. of Operational Research*, 239(3):636–643, 2014.