Locating Evacuation Centers Optimally in Path and Cycle Networks

Robert Benkoczi
Department of Mathematics and Computer Science, University of Lethbridge, Canada

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

Yuya Higashikawa
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Tsunehiko Kameda[1]
School of Computing Science, Simon Fraser University, Burnaby, Canada

Naoki Katoh
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Junichi Teruyama
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Abstract

We present dynamic flow algorithms to solve the *k-sink problem* whose aim is to locate k sinks (evacuation centers) in such a way that the evacuation time of the last evacuee is minimized. In the *confluent model*, the evacuees originating from or passing through a vertex must evacuate to the same sink, and most known results on the k-sink problem adopt the confluent model. When the edge capacities are uniform (resp. general), our algorithms for non-confluent flow in the path networks run in $O(n+k^2 \log^2 n)$ (resp. $O(n \log n+k^2 \log^5 n)$) time, where n is the number of vertices. Our algorithms for cycle networks run in $O(k^2 n \log^2 n)$ (resp. $O(k^2 n \log^5 n)$) time, when the edge capacities are uniform (resp. general).

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Efficient algorithms, facility location, minmax sink, evacuation problem, dynamic flow in network

Digital Object Identifier 10.4230/OASIcs.ATMOS.2021.13

Funding *Robert Benkoczi*: Partially supported by a Discovery Grant from NSERC of Canada.
Binay Bhattacharya: Partially supported by a Discovery Grant from NSERC of Canada.
Yuya Higashikawa: Partially supported by JSPS KAKENHI Grant Number 20K19746 and JSPS KAKENHI Grant Number 19H04068.
Naoki Katoh: Partially supported by JSPS KAKENHI Grant Number 19H04068.
Junichi Teruyama: Partially supported by JSPS KAKENHI Grant Number 19H04068.

1 Introduction

Ford and Fulkerson [8] introduced the concept of *dynamic flow* which models movement of commodities in a network. Each vertex is assigned some initial amount of supply, and each edge has a capacity, which limits the rate of commodity flow into it, and the transit time to traverse it. Once on an edge, the flow front travels at a constant speed either to a sink, or to the vertex at the other end of the edge if there is no sink on the edge. *Congestion* is said to occur when supplies cannot flow continuously but must wait at some vertex to enter an outgoing edge, and congestion complicates the analysis.

[1] Corresponding author.

One variant of the dynamic flow problem is the *quickest transshipment* problem, where the source vertices have given amounts of supplies and the sink vertices have specified demands. The problem is to send exactly the right amount of commodity out of the sources into the sinks in minimum overall time. Hoppe and Tardos [13] provided a polynomial time algorithm for this problem in the case where the transit times are integral. However, the complexity of their algorithm is impractically high. Finding a practical polynomial time solution to this problem is still open. The interested reader is referred to a survey paper by Skutella [17].

The *k-sink problem* is another variant of the dynamic flow problem, whose aim is to locate a set of k sinks that accepts the *evacuees* in minimum time [10, 15]. Each source vertex has an initial number of evacuees, and each sink has infinite demand, namely it can receive an arbitrary number of evacuees with no capacity constraint. Evacuation starts simultaneously from all vertices. For general graphs, even the 1-sink problem is NP-hard [9, 14]. We thus address simple networks (path and cycle) in this paper.

The existing solutions to the evacuation problem impose the condition that all evacuees starting at or passing through a vertex must evacuate to the same sink. This can be justified by the fact that posting "This way out" signs at each vertex, directing the evacuees to a single exit, will avoid confusion. Such flow is called *confluent* flow. Adopting the confluence restriction, Arumugam et al. [2] showed that the k-sink problem for path networks with general edge capacities can be solved in $O(kn \log^2 n)$ time, where n is the number of vertices. A path network can model a corridor in a building, an aisle in an airplane, a street, etc. As for the uniform edge capacity model, Higashikawa et al. [11] then proposed an $O(kn)$ time algorithm. More recently, Bhattacharya et al. [4] improved it to $O(\min\{n+k^2 \log^2 n, n \log n\})$ time, and also presented an algorithm for the general edge capacity model that runs in $O(\min\{n \log n+k^2 \log^4 n, n \log^3 n\})$ time. These improvements were achieved by moving from dynamic programming based approach to parametric search based methods. A recent comprehensive survey on evacuation problems can be found in [12].

In this paper we consider *non-confluent* flow solution to the evacuation problem on path and cycle networks. This means that the evacuees from a vertex can move in two opposite directions. It can be practical, if each potential evacuee is given the exit number beforehand, so that he/she knows exactly which exit to take in case of emergency. We will treat each evacuee as if he/she was a tiny particle with a very small weight. This paper presents an algorithm that runs in $O(n+k^2 \log^2 n)$ (resp. $O(n \log n+k^2 \log^5 n)$) time for the uniform (resp. general) edge capacity model. Benkoczi and Das [3, 7] solve the k-sink problem for cycle networks for confluent flows, which run in time $O(n \log n)$ (resp. $O(n \log^3 n)$) when the edge capacities are uniform (resp. general).

This paper is organized as follows. After preliminaries in Sec. 2, we discuss the uniform capacity model in Secs. 3 and Sec. 4, where we deal with feasibility testing and optimization, respectively. Sec. 5 then discusses the general capacity model, and Sec. 6 extends the results to cycle networks.

2 Preliminaries

2.1 Definitions

Let $P(V, E)$ be a path network with the set of vertices $V = \{1, 2, \ldots, n\}$, arranged from left to right in this order. For each i $(1 \leq i \leq n-1)$, there is an edge $e_i = (i, i+1) \in E$, which does not include its end vertices. For each vertex $i \in V$, let $w_i \in \mathbb{Z}_+$ denote its *weight*, which is the initial number of evacuees at vertex i, and for each $e_i \in E$, let $c(i, i+1)$ denote its *capacity*, which limits the number of evacuees who can enter e_i from i or $i+1$ per unit time.

By $a \in P$ we denote the fact that point a lies on P, either at a vertex or on an edge. For any two points $a, b \in P$, we write $a \prec b$ to mean that a lies to the left of b, and $a \preceq b$ means $a \prec b$ or $a = b$. The minimum capacity between a and b is denoted by $c(a, b)$. Let l_i denote the length of edge e_i. We use $d(a, b)$ to denote the distance between a and b, which is the sum of the edge lengths. If a and/or b lies on an edge, its prorated length is used.

For a pair of values or positions x and y, $[x, y]$ denotes the range or interval from x to y, inclusive, while $(x, y]$ (resp. $[x, y)$) denotes the range from x to y, excluding x (resp. y). For $a \preceq b$, $P[a, b]$ (resp. $P(a, b]$, $P[a, b)$) denotes the subpath of P from a to b with the above interpretation of the range. The set of vertices on $P[a, b]$ (resp. $P(a, b]$, $P[a, b)$) are denoted by $V[a, b]$ (resp. $V(a, b]$, $V[a, b)$). For a technical reason, we define v^+ (resp. v^-) to be an imaginary vertex such that $v \prec v^+$, $d(v, v^+)$=0, and $c(v, v^+)$=$c(v, v{+}1)$ (resp. $v^- \prec v$, $d(v^-, v)$=0, and $c(v^-, v)$=$c(v{-}1, v)$). Let τ denote the unit distance travel time, so that for a evacuee to travel from a to b, without encountering congestion, requires $d(a, b)\tau$ units of time.

Our model assumes that evacuation starts at the same time from every vertex.

▶ **Definition 1.** *Let* $W[i, j] \triangleq \sum_{h \in V[i,j]} w_h$*, and for* $h, i, j \in V$ *such that* $i \preceq h \preceq j$ *define*

$$f_L^{[i,\cdot]}(x, h) \triangleq \begin{cases} d(h, x)\tau + W[i, h]/c(h, x) & \text{for } x \succ h \\ 0 & \text{for } x \preceq h, \end{cases} \tag{1}$$

$$f_R^{[\cdot,j]}(x, h) \triangleq \begin{cases} d(x, h)\tau + W[h, j]/c(x, h) & \text{for } x \prec h \\ 0 & \text{for } x \succeq h. \end{cases} \tag{2}$$

Intuitively, $f_L^{[i,\cdot]}(x, h)$ is the evacuation time of all evacuees on $P[i, h]$ to point $x \succeq h$, assuming that all of them were at vertex h initially, and the flow from $P[h{+}1, x]$ to x does not interfere with it. Similarly $f_R^{[\cdot,j]}(x, h)$ is the evacuation time of all evacuees on $P[h, j]$ to $x \preceq h$, assuming that all of them were at vertex h initially, and the flow from $P[x, h{-}1]$ to x does not interfere with their flow. We now define their upper envelopes.

▶ **Definition 2.** *For* $i, j \in V$*, define*

$$\Theta_L^{[i,\cdot]}(x) \triangleq \max_{v \in V[i,x)} \left\{ f_L^{[i,\cdot]}(x, v) \right\} = f_L^{[i,\cdot]}(x, v_x^*), \tag{3}$$

$$\Theta_R^{[\cdot,j]}(x) \triangleq \max_{v \in V(x,j]} \left\{ f_R^{[\cdot,j]}(x, v) \right\} = f_R^{[\cdot,j]}(x, v_x^*). \tag{4}$$

The rightmost (resp. leftmost) vertex v_x^* *satisfying Eq.* (3) *(resp. Eq.* (4)*) is called the* L-critical vertex *(resp.* R-critical vertex*) for* $P[i, x]$ *(resp.* $P[x, j]$*) w.r.t.* x*, and is denoted by* $\rho_L^{[i,\cdot]}(x)$ *(resp.* $\rho_R^{[\cdot,j]}(x)$*).*

▶ **Lemma 3** ([12])**.** *For any point* $x \succ i$ *(resp.* $x \prec i$*),* $\Theta_L^{[i,\cdot]}(x)$ *(resp.* $\Theta_R^{[\cdot,j]}(x)$*) is the evacuation time for all evacuees on* $P[i, x)$ *(resp.* $P(x, j]$*) to* x*.*

We thus refer to $\Theta_L^{[i,\cdot]}(x)$ (resp. $\Theta_R^{[\cdot,j]}(x)$) as the *L-time* (resp. *R-time*) for $P[i, x]$ (resp. $P[x, j]$) at x.

▶ **Definition 4.** *An instance* P *of a path network is said to be* (λ, k)-feasible *or just* λ-feasible*, if* k *sinks can be placed on it so that every evacuee can evacuate to a sink within time* λ*. The* λ-feasibility test *decides if the given instance* P *is* (λ, k)*-feasible.*

2.2 Megiddo's lemma

We shall apply the following lemma implied by Megiddo's observation [16].

▶ **Lemma 5.** *Let $cmp(n)$ be the number of comparisons made with λ in a λ-feasibility test, $t(n)$ be the time needed to generate a λ value to be tested, $f(n)$ be the time complexity of the λ-feasibility test, and $h(n)$ be the time required by all other operations. Then the optimal solution to the k facility location problem can be found in time*

$$O(h(n) + cmp(n)\{f(n) + t(n)\}). \tag{5}$$

For a clear exposition of the idea behind this lemma, the reader is referred to Agarwal and Sharir [1]. Intuitively, we replace each comparison with λ in the λ-feasibility test by a comparison with λ^*, where λ^* is the optimal solution. Note that a feasibility test is actually a comparison of some value λ with λ^*, and it succeeds (resp. fails) if $\lambda \geq \lambda^*$ (resp. $\lambda < \lambda^*$). To determine λ^*, we perform successive λ-feasibility tests, using judiciously chosen λ values.

2.3 Review of CUE tree and CV tree

The *critical vertex tree* (*CV tree*) was introduced in [5], and the *capacity and upper envelope tree* (*CUE tree*) was introduced in [4]. They are balanced binary trees built over path P. Since they play an important role in this paper, we briefly review them for completeness.

The leaf nodes[2] of the CUE-tree, denoted by $\mathcal{T}$, are the vertices of P. See Fig. 1, for example. For node u of $\mathcal{T}$, let $l(u)$ (resp. $r(u)$) denote the leftmost (resp. rightmost) vertex

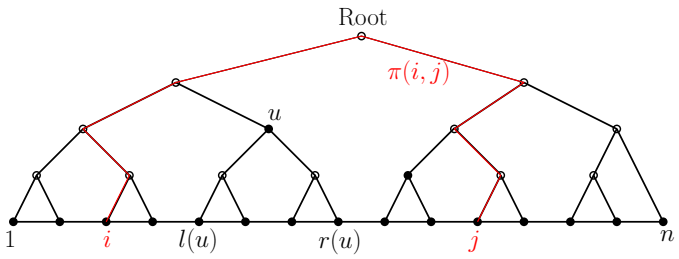

Figure 1 The structure of a CUE tree $\mathcal{T}$.

of P that belongs to subtree $\mathcal{T}(u)$, rooted at u. We say that u *spans* subpath $P[l(u), r(u)]$, whose vertex set is denoted by $V(u)$. Let $N[i,j]$ denote the set of nodes spanning the maximal subpaths of $P[i,j]$. Each node in $N[i,j]$ either lies on the path $\pi(i,j)$ from i to j or is a child of a node on $\pi(i,j)$. Each node u of $\mathcal{T}$ stores

(i) $l(u)$ and $r(u)$,
(ii) capacity $c(i,j)$,
(iii) four 1-dimensional arrays, which are described below.

Given $i, j \in V$, consider any node $u \in N[i, j-1]$. The L-time at $j \succ r(u)$ for the supplies from $V(u)$ is given by

$$\max_{h \in V(u)} \left\{ d(h, r(u))\tau + \frac{W[i,h]}{\min\{c(h, r(u)^+), c\}} \right\} + d(r(u), j)\tau, \tag{6}$$

where $c = c(r(u), j)$. We rewrite the first term of Eq. (6) as

$$\max_{h \in V(u)} \left\{ d(h, r(u))\tau + \max\left(\frac{W[i,h]}{c(h, r(u)^+)}, \frac{W[i,h]}{c} \right) \right\}, \tag{7}$$

[2] We use the term "node" here to distinguish it from the vertices on P.

which obviously equals

$$\max\{\Theta_L^w(W,u), \tilde{\Theta}_L^c(c,u)+W/c\}, \tag{8}$$

where

$$\Theta_L^w(W,u) \triangleq \max_{h\in V(u)} \left\{ d(h,r(u))\tau + \frac{W}{c(h,r(u)^+)} + \frac{W[l(u),h]}{c(h,r(u)^+)} \right\} \tag{9}$$

$$\tilde{\Theta}_L^c(c,u) \triangleq \max_{h\in V(u)} \left\{ d(h,r(u))\tau + \frac{W[l(u),h]}{c} \right\}. \tag{10}$$

Here W and c are unknowns at the time of constructing $\mathcal{T}$ during preprocessing. Note that $\Theta_L^w(W,u)$ is a piecewise linear function in W, consisting of $O(|V(u)|)$ linear segments. We need a method by which to compute the inflection points $(W,\Theta_L^w(W,u))$ of the upper envelope (9). Let $A_L^w(u)$ denote the 1-dimensional *left weight array* stored at node u of $\mathcal{T}$. It contains the W components of the inflection points $(W,\Theta_L^w(W,u))$ in the increasing order of W. For a technical reason, we consider $(0,\Theta_L^w(0,u))$ and (∞,∞) as inflection points of $(W,\Theta_L^w(W,u))$. Let $A_L^c(u)$ denote the 1-dimensional *left capacity array* stored at node u. For a technical reason, we consider $(0,\infty)$ and $(\infty,\Theta_L^c(\infty,u))$ as inflection points of $\Theta_L^c(c,u)$. The size of $A_L^w(u)$ and $A_L^c(u)$ is clearly $O(|\mathcal{T}(u)|)$.

Suppose we want to compute the L-time of $V[i,j-1]$ at j. To this end, we calculate the contributions from subtree $\mathcal{T}(u)$, for each $u \in N[i,j-1]$ separately, and find their maximum. As we saw above, the contribution from each subtree $\mathcal{T}(u)$ is given by $\max\{\Theta_L^w(W,u), \tilde{\Theta}_L^c(c,u)+W/c\}$, where $W = W[i,l(u)-1]$ and $c = c(r(u),j)$.

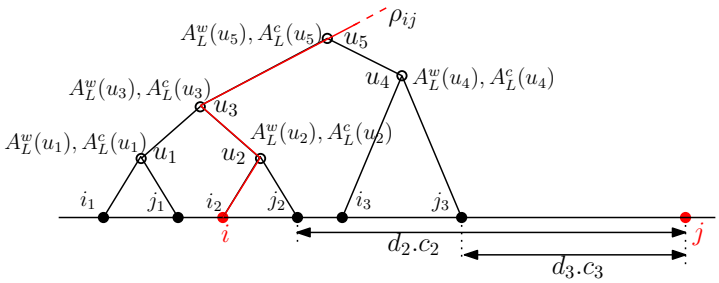

Figure 2 Weight arrays $A_L^w(u)$ and capacity arrays $A_L^c(u)$. $A_R^w(u)$ and $A_R^c(u)$ are not shown in this figure.

▶ **Example 6.** Let $W[u]$ denote the total weights of $V[u]$, and let $u = u_4$ in Fig. 2, for example. Then we search in array $A_L^w(u_4)$ with the search key $W = W[u_2]$, to find the two successive inflection points between which W falls, and obtain the time $\Theta_L^w(W[u_2],u_4)$ by interpolating between the two points. Adding $d_3\tau$ to it, we find the L-time of $V(i,r(u_4))$ at j, which is just one candidate for the true L-time of $V[i,j-1]$ at j. We repeat this for all $u \in N[i,j-1]$. ⌟

As for $\tilde{\Theta}_L^c(c,u)$, it is useful to consider it as a linear function in $1/c$, so that it is also piecewise linear, consisting of $O(|V(u)|)$ linear segments. At u, we store the 1-dimensional *capacity array* $A_L^c(u)$ in the increasing order of c, containing the c components of the inflection points $(c,\tilde{\Theta}_L^c(c,u))$. After searching for $c(r(u_4),j)$ in $A_L^c(u_4)$ in Fig. 2, for example, and interpolation, we need to compute $\tilde{\Theta}_L^c(c(r(u_4),j),u_4)+W[i_2,j_2]/c(r(u_4),j)$ to arrive at another candidate L-time at j.

Symmetrically to (9) and (10), we also define

$$\Theta_R^w(W,u) \triangleq \max_{h\in V(u)} \left\{ d(l(u),h)\tau + \frac{W}{c(l(u)^-,h)} + \frac{W[h,r(u)]}{c(l(u)^-,h)} \right\} \tag{11}$$

$$\tilde{\Theta}_R^c(c,u) \triangleq \max_{h\in V(u)} \left\{ d(l(u),h)\tau + \frac{W[h,r(u)]}{c} \right\}. \tag{12}$$

We construct and store at node u the *right weight array* $A_R^w(u)$ and *right capacity array* $A_R^c(u)$, based on (11) and (12), respectively. Thus $A_R^w(u)$ (resp. $A_R^c(u)$) is left-right symmetric to $A_L^w(u)$ (resp. $A_L^c(u)$).

The *CV tree* [5] is a simplified version of the CUE tree, which is useful for the uniform capacity model. Instead of data (ii) and (iii), node u stores the L-critical (resp. R-critical) vertex $\rho_L^{[l(u),\cdot]}(r(u)^+)$ (resp. $\rho_R^{[\cdot,r(u)]}(l(u)^-)$).

▶ **Lemma 7** ([4, 5]). *The CV tree (resp. CUE tree) can be constructed in $O(n)$ (resp. $O(n\log n)$) time.*

▶ **Lemma 8** ([5]). *Assume CV tree $\mathcal{T}$ is available, and let $i<j$. For the uniform edge capacity model, we can compute*
(a) *$\rho_L^{[i,\cdot]}(j)$ and $\rho_R^{[\cdot,j]}(i)$ in $O(\log n)$ time.*
(b) *$\Theta_L^{[i,\cdot]}(x)$, $\Theta_R^{[\cdot,j]}(x)$, and $\Theta^{[i,j]}(x)$ in $O(\log n)$ time for any point $x\in P[i,j]$.*

▶ **Lemma 9** ([4]). *Assume CUE tree $\mathcal{T}$ is available, and let $i<j$. For the general edge capacity model, we can compute*
(a) *$\rho_L^{[i,\cdot]}(j)$ and $\rho_R^{[\cdot,j]}(i)$ in $O(\log^2 n)$ time.*
(b) *$\Theta_L^{[i,\cdot]}(x)$, $\Theta_R^{[\cdot,j]}(x)$, and $\Theta^{[i,j]}(x)$ in $O(\log^2 n)$ time for any point $x\in P[i,j]$.*

Using fractional cascading, we can reduce the time complexity in Lemma 9 to $O(\log n)$.

2.4 Strategy

Let k (≥ 2) be the number of sinks to be placed, since if $k=1$ the flows are always confluent. We want to compute $\{s_i, b_i, \alpha_i \mid 1\leq i\leq k\}$ that minimize the evacuation time, where s_i is the location of the i^{th} sink from the left, b_i is the rightmost vertex from which at least some evacuees move left to s_i, and is called the *boundary vertex* for s_i. α_i evacuees ($0<\alpha_i\leq w_{b_i}$) evacuate left from b_i to s_i. Let $\bar{\alpha}_i \triangleq w_{b_i}-\alpha_i$, if $\alpha_i < w_{b_i}$.

Path partitioning idea: Imagine that we have performed the λ^*-feasibility test. The result may look like Fig. 3, where the dots and small circles represent vertices. Each triangle represents a sink, a red triangle represents a sink placed at a vertex, and a dot that is not a sink indicates a boundary vertex whose evacuees are not split. Let us remove the edges

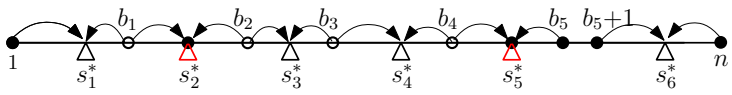

Figure 3 $\{s_i^*\}$ are optimal sinks and $\{b_i\}$ are boundary vertices between adjacent sinks.

carrying no flow, which are incident on non-split boundary vertices, if any, and then divide each sink that lies on a vertex into two sinks, one of which is attached to its left incident edge, and the other to its right incident edge. Then each connected subpath would be one of the four types shown in Fig. 4, where there is at least one sink in each subpath. For example,

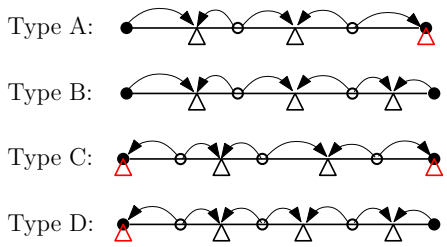

Figure 4 Four types of subpaths.

a subpath of Type A starts with vertex 1 or a vertex that is the right neighbor of a non-split boundary vertex and ends with a sink at a vertex. Our optimization algorithms locate sinks in each subpath in the optimal way. The optimal evacuation time is given by the subpath with the maximum evacuation time.

3 Feasibility test

Given a value (evacuation time) λ, starting from the left end of P, we identify the rightmost point s_1 such that all evacuees on $P[1, s_1)$ can evacuate right to s_1 within time λ. We then determine the *boundary vertex* for s_1, denoted by b_1, such that all evacuees on $P(s_1, b_1{-}1]$ and $\alpha_1 (0 < \alpha_1 \leq w_{b_1})$ evacuees from b_1 can evacuate left to s_1 within time λ, and b_1 is the rightmost such vertex. We repeat this for the remaining part $P[b_1, n]$ of P, with the weight of b_1 reduced to $\bar{\alpha}_1 = w_{b_1} - \alpha_1$, if $\alpha_1 < w_{b_1}$, and detaching $P[1, b_1]$ from P if $\alpha_1 = w_{b_1}$. It is clear that the given instance P is λ-feasible, if and only if the end vertex n of P is reached before no more than k sinks are introduced this way. We present this approach later as Algorithm **FTest**, after introducing its building blocks formally.

3.1 Finding maximal λ-covered subpaths

We say that vertex i is λ-*covered* by sink s, if $\Theta_L^{[i,\cdot]}(s) \leq \lambda$ or $\Theta_R^{[\cdot,i]}(s) \leq \lambda$. A subpath is said to be λ-*covered*, if every vertex on it is λ-covered by a sink.

3.1.1 Finding the next sink

Given a λ value, assume that we have introduced sinks, $\{s_1, \ldots, s_{i-1}\}$, and the associated boundary vertices, $\{b_1, b_2, \ldots, b_{i-1}\}$, so far for some $i \geq 1$, and we want to locate the new sink s_i. Assume further that each sink λ-covers a maximal subpath, and lies at the rightmost position possible. As the initial condition, we set $\bar{\alpha}_0 = w_1$, which implies that the amount w_1 must be sent to s_1. For $i \geq 2$, α_{i-1} $(0 < \alpha_{i-1} \leq w_{b_{i-1}})$ evacuees travel left to s_{i-1}. If $\alpha_{i-1} < w_{b_{i-1}}$,[3] then $\bar{\alpha}_{i-1} \triangleq w_{b_{i-1}} - \alpha_{i-1}$ (> 0) evacuees must travel right to s_i. Let $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(x)$,[4] where $x \succeq b_{i-1}$, denote the evacuation time for the evacuees from $P[b_{i-1}, x)$ at x, with the weight of b_{i-1} changed to $\bar{\alpha}_{i-1}$. For the purpose of testing λ-feasibility, we want to find a sink s_i, farthest from b_{i-1} on its right, such that $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(s_i) \leq \lambda$. Since $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(x)$ monotonically increases as we move x to the right, we can perform binary search to determine adjacent vertices $v, v{+}1 \in V$ such that $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v) < \lambda$ and $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v{+}1) \geq \lambda$. See Fig. 5. Then we can locate sink s_i on subpath $P[v, v{+}1]$ in constant time.[5]

[3] If not, then $b_{i-1}{+}1$ is like v_1. See $b_5{+}1$ in Fig. 3.
[4] It implicitly depends on λ, which indicated by the tilde on Θ.
[5] Note that v is included, because $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) \geq \lambda$ is possible. See Lemma 11.

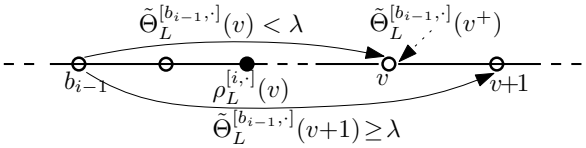

Figure 5 Looking for the next sink s_i.

▶ **Observation 10.** In the uniform edge capacity model, for a point $x \succ b_{i-1}$, the L-critical vertex $\rho_L^{[b_{i-1},\cdot]}(x)$ for $P[b_{i-1}, x)$ w.r.t. x does not depend on $\bar{\alpha}_{i-1}$ as long as $\bar{\alpha}_{i-1} > 0$, and $\rho_L^{[b_{i-1},\cdot]}(x)$ satisfies

$$\tilde{\Theta}_L^{[b_{i-1},\cdot]}(x) = d(\rho_L^{[b_{i-1},\cdot]}(x), x)\tau + \frac{W[b_{i-1}, \rho_L^{[b_{i-1},\cdot]}(x)] - \alpha_{i-1}}{c} \tag{13}$$

$$= d(\rho_L^{[b_{i-1},\cdot]}(x), x)\tau + \frac{W[b_{i-1}{+}1, \rho_L^{[b_{i-1},\cdot]}(x)] + \bar{\alpha}_{i-1}}{c}, \tag{14}$$

where we define $W[b_{i-1}{+}1, b_{i-1}] = 0$. ⌟

▶ **Lemma 11.** *For a given λ, let $v \in V$ satisfy $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v) < \lambda$ and $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) = \tilde{\Theta}_L^{[b_{i-1},\cdot]}(v{+}1) - l_v\tau \geq \lambda$.*[6] *Then v is the rightmost possible position for sink s_i.*

Proof. It is clear that if $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) > \lambda$, then s_i must be placed at v. So consider the case where $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) = \lambda$. In this case, the evacuation time will be $> \lambda$ at any location $\succ v^+$. We could place the sink at the imaginary location v^+, but we might as well place it at v, since there is no difference in the coverage on the left or right side, if s_i is placed at v^+ or at v. ◀

When the condition of the above lemma holds, s_i is said to be *vertex-bound*, *VB* for short, to v for λ. The L-time jumps beyond λ if the sink is placed at any finite distance (> 0) to the right of v.

▶ **Lemma 12.** *Procedure* **NxtSink**$(\lambda, a, \overline{\alpha})$*, presented below, finds the next sink correctly, and if the CV tree $\mathcal{T}$ is available, then it runs in $O(\log n)$ time, performing $O(\log n)$ comparisons with λ.*

Proof. The correctness follows from the above discussions. The complexity follows from Lemma 8, since the portion of the split weight to be λ-covered by the next sink is known. ◀

3.1.2 Finding the boundary vertex for sink s_i

We now look for s_i's boundary vertex b_i, as illustrated in Fig. 6.

▶ **Lemma 13.** *Given sink $s_i \in P(v, v{+}1]$, let b be the leftmost vertex such that $\Theta_R^{[\cdot,b]}(s_i) \geq \lambda$. If $d(s_i, b)\tau > \lambda$, then we must have $b_i = b{-}1$ and all the w_{b-1} evacuees can evacuate to s_i within time λ. Otherwise, we have $b_i = b$, and a portion of w_b can evacuate to s_i within time λ.*

[6] This implies that $\rho_L^{[b_{i-1},\cdot]}(v^+) = v$.

Procedure NxtSink$(\lambda, a, \overline{\alpha})$.

Input : a (vertex with weight $\overline{\alpha}$ $(0 < \overline{\alpha} \leq w_a)$), λ (limit on L-time)
Output : v, $d(v,s)$, $s \in P[v, v{+}1]$ (next sink)

1 **if** $\tilde{\Theta}_L^{[a,\cdot]}(n) \leq \lambda$ **then**
2 $\quad$ $s \leftarrow n$, $v \leftarrow n{-}1$, and $d(v,s) \leftarrow l_{n-1}$ **stop**.
3 **end**
4 Using binary search, find $h \in [a, \ldots, n{-}1]$ such that $\tilde{\Theta}_L^{[a,\cdot]}(h) < \lambda$ and $\tilde{\Theta}_L^{[a,\cdot]}(h{+}1) \geq \lambda$;
5 **if** $\tilde{\Theta}_L^{[a,\cdot]}(h^+) = \tilde{\Theta}_L^{[a,\cdot]}(h{+}1) - l_h\tau \geq \lambda$ **then**
6 $\quad$ $d(h,s) \leftarrow 0$; `// s is VB to h.`
7 **else**
8 $\quad$ $d(h,s) \leftarrow \{\lambda - \tilde{\Theta}_L^{[a,\cdot]}(h^+)\}/\tau$;
9 **end**
10 $s \leftarrow$ point at distance $d(h,s)$ to the right of v ;
11 $v \leftarrow h$.

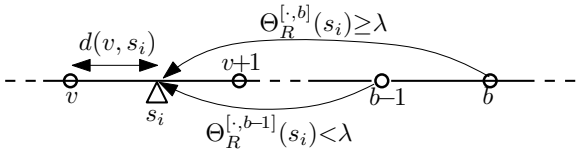

Figure 6 Finding the boundary vertex b_i for sink s_i.

Proof. If $d(s_i, b)\tau > \lambda$, then clearly it takes more than λ time for the first evacuee from b to arrive at s.

Otherwise, there are two cases to consider. If $\frac{W[h,b_i-1]}{c} < d(h,b_i)\tau$, where $h = \rho_R^{[\cdot,b-1]}$, (b_i is the R-critical vertex w.r.t. s_i), then at least one evacuee from b_i can arrive at s_i within λ. If $\frac{W[h,b_i-1]}{c} \geq d(h,b_i)\tau$ (b_i is not the R-critical vertex w.r.t. s_i), on the other hand, since $\Theta_R^{[\cdot,b-1]}(s_i) < \lambda$, the first arrival from b can reach s_i within λ, hence $b_i = b$. ◀

If $b_i = b{-}1$ in the above lemma, we call it the *separator vertex* for the current subpath. If $b_i = b$, on the other hand, we look for the split portion α_i $(0{<}\alpha_i{\leq}w_b)$ by setting

$$\Theta_R^{[\cdot,b-1]}(s_i) + \alpha_i/c = \lambda,$$

$$\text{or} \quad d(s_i, \rho_R^{[\cdot,b-1]}(s_i))\tau + \frac{W[\rho_R^{[\cdot,b-1]}(s_i), b{-}1] + \alpha_i}{c} = \lambda. \tag{15}$$

We now solve (15) for α_i, which yields

$$\alpha_i = \{\lambda{-}d(s_i, \rho_R^{[\cdot,b-1]}(s_i))\tau\}c - W[\rho_R^{[\cdot,b-1]}(s_i), b{-}1]. \tag{16}$$

If b is the R-critical vertex w.r.t. s_i, then we solve $d(s_i,b)\tau{+}\alpha_i/c{=}\lambda$, instead of (15).

Procedure **R-Bnd**(λ, s_i, d), given below, computes the boundary vertex b_i for s_i and also α_i.

Since Step 4 makes $O(\log n)$ probes, we have

▶ **Lemma 14.** *If the CV tree $\mathcal{T}$ is available and the position of s_i is known,* **R-Bnd** *computes the boundary vertex for a sink in $O(\log n)$ time by comparing $O(\log n)$ values with λ.*

Procedure R-Bnd(λ, s_i, d).

Input : λ, s_i, d // $s_i \in P[v, v{+}1)$ and $d = d(v, s_i)$.
Output : b_i, α_i $(0 < \alpha_i \le w_{b_i})$
1 **if** $\Theta_R^{[\cdot,n]}(s_i) \le \lambda$ **then**
2 Set $b_i \leftarrow n$ and $\alpha_i \leftarrow w_n$
3 **else**
4 Using binary search, find vertex b such that $\Theta_R^{[\cdot,b-1]}(s_i) < \lambda$ and $\Theta_R^{[\cdot,b]}(s_i) \ge \lambda$;
5 **if** $d(s_i, b)\tau > \lambda$ **then**
6 $b_i \leftarrow b{-}1$, $\alpha_i \leftarrow w_{b-1}$; // b_i is a separator vertex.
7 **else**
8 $b_i \leftarrow b$, and compute α_i, using (16) with $b = b_i$
9 **end**
10 **end**

3.2 Feasibility test algorithm

Algorithm **FTest** below presents our feasibility test as a pseudo code. It makes $O(k)$ calls to **NxtSink** and **R-Bnd**, which are the most time consuming operations.

Lemmas 12 and 14 imply

▶ **Lemma 15.** *Algorithm* **FTest** *makes* $cmp(n) = O(k \log n)$ *comparisons with* λ.

▶ **Lemma 16.** *If the CV tree* $\mathcal{T}$ *is available, then Algorithm* **FTest** *decides* λ*-feasibility for a given* λ *in* $f(n) = O(k \log n)$ *time.*

Proof. When $i = 1$ in the **while** loop, **NxtSink**$(\lambda, b_{i-1}, \bar{\alpha}_{i-1})$=**NxtSink**$(\lambda, 1, w_1)$ in Step 3, and it generates the exact distance $d(v, s_1)$ in $O(\log n)$ time by Lemma 12. This distance is fed to **R-Bnd**(λ, s_1, d) in Step 5 as $d = d(v, s_1)$, and it generates the exact split portion α_1 in $O(\log n)$ time by Lemma 14. We can now compute $\bar{\alpha}_1 = w_{b_1} - \alpha_1$ as an input to the second invocation of **NxtSink**. The lemma follows by repeating this argument k times. ◀

4 Optimization for the uniform capacity model

In applying Lemma 5, we now know that $cmp(n) = O(k \log n)$ and $f(n) = O(k \log n)$ from Lemmas 15 and 16, respectively. Thus the remaining problem is to find $t(n)$, which is the time needed to identify the next λ value to be tested for feasibility. Suppose that we have located the first $i{-}1$ sinks $\{s_1, s_2, \ldots, s_{i-1}\}$ on edges, where $i \le k$, based on the current upper bound $\overline{\lambda}$. They are obtained as a result of the last successful feasibility test. Note that a $\overline{\lambda}$-feasibility test has already been performed, because it is how $\overline{\lambda}$ was updated to the current value.

4.1 Finding the next sink in optimal solution

As a result of the last successful feasibility test, the L-time and R-time of each sink s_h $(h \le i{-}1)$ equal $\overline{\lambda}$, and we have the sink locations $\{s_h \mid 1 \le h \le i{-}1\}$ on edges and the split portions $\{\alpha_h \mid 1 \le h \le i{-}1\}$ of the boundary vertices $\{b_h \mid 1 \le h \le i{-}1\}$. See Fig. 7. Since we know $\overline{\alpha}_{i-1}$, based on it, we can compute the L-time $\lambda_v = \tilde{\Theta}_L^{[b_{i-1},\cdot]}(v)$ at vertex $v \succ b_{i-1}$, being probed using binary search as a candidate for sink s_i.

Algorithm 1 **FTest**.

Input : P, λ, k
Output : `Feasible/Infeasible`, updated $\underline{\lambda}$ or $\overline{\lambda}$, $\{s_i, b_i \mid 1 \le i \le k\}$

$i \leftarrow 1$; $b_0 \leftarrow 1$; $\bar{\alpha}_0 \leftarrow w_1$;
while $[i < k] \wedge$ [Vertex n is not λ-*covered* by a sink] **do**
  By invoking **NxtSink**$(\lambda, b_{i-1}, \bar{\alpha}_{i-1})$, find the next sink s_i and $d(v, s_i)$, where $v \preceq s_i \prec v{+}1$;
  Set $d \leftarrow d(v, s_i)$; `// If` $d(v, s_i) = 0$`, then` s_i `is VB to` v `w.r.t.` λ.
  Run Procedure **R-Bnd**(λ, s_i, d); `// Generate` b_i `and` α_i.
  if $\alpha_i = w_{b_i}$ **then**
    $b_i \leftarrow b_i{+}1$; `// The next subpath should start from updated` b_i.
  else
    $\bar{\alpha}_i \leftarrow w_{b_i} - \alpha_i$;
  end
  $i \leftarrow i{+}1$
end
if *vertex* n *is* λ-covered *by a sink* **then**
  $\overline{\lambda} \leftarrow \lambda$;
  Output `Feasible`
else
  $\underline{\lambda} \leftarrow \lambda$;
  Output `Infeasible`.
end
Output $\{s_i, b_i \mid 1 \le i \le k\}$, $\overline{\lambda}$ and $\underline{\lambda}$.

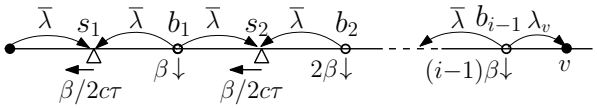

Figure 7 Equalization. Reduction in α_h is $h\beta$ for $1 \le h \le i{-}1$.

Whenever we probe such a vertex v, we need to compute the "equalized" evacuation time at all the sinks $\{s_h \mid 1 \le h \le i{-}1\}$,[7] including the candidate sink s_i that may be placed at v. We may assume that $\lambda_v < \overline{\lambda}$, since otherwise, the equalized time will be larger than $\overline{\lambda}$, and we would already know the outcome of its feasibility test. We thus have a "slack" of $\overline{\lambda}{-}\lambda_v$, which means that the L-time and R-time of each s_h $(1 \le h \le i{-}1)$ can be reduced to remove this slack. For equalization, we move sinks $\{s_h \mid 1 \le h \le i{-}1\}$ left, and also reduce the split portions $\{\alpha_h \mid 1 \le h \le i{-}1\}$ to make all the L-time and R-time at all the sinks the same, which should equal the minimum evacuation time for subpath that we are processing, if s_i is placed at v (Type A or C).

▶ **Lemma 17.** *Let* $\underline{\lambda}$ *and* $\overline{\lambda}$ *be the current bounds, and let* $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ *be the non-VB sinks and split boundary vertices, based on the current* $\overline{\lambda}$ *that resulted from the last successful feasibility test. If* λ *is moved within* $(\underline{\lambda}, \overline{\lambda})$*, then*

(a) *Each sink* s_h $(1 \le h \le i{-}1)$ *moves on the same edge.*
(b) *Each boundary vertex* b_h $(1 \le h \le i{-}1)$ *does not change.*

[7] If no sink has been introduced ($i{=}1$), this step is not needed.

Proof. **(a)** Let $s_h \in (u, u{+}1)$. In the process of determining that s_h is not VB, we would have tested the feasibility of $\tilde{\Theta}_L^{[b_{h-1},\cdot]}(u)$ and $\tilde{\Theta}_L^{[b_{h-1},\cdot]}(u{+}1)$, and the former (resp. latter) must have failed (resp. succeeded). Thus the bounds were updated by $\underline{\lambda} \leftarrow \max\{\underline{\lambda}, \tilde{\Theta}_L^{[b_{h-1},\cdot]}(u)\}$ and $\overline{\lambda} \leftarrow \min\{\overline{\lambda}, \tilde{\Theta}_L^{[b_{h-1},\cdot]}(u{+}1)\}$. This implies that s_h cannot reach u or $u{+}1$ if $\lambda \in (\underline{\lambda}, \overline{\lambda})$, and will stay on edge $(u, u{+}1)$.

(b) Assume that changing λ within $(\underline{\lambda}, \overline{\lambda})$ caused $\alpha_h = 0$ or $\alpha_h = w_{b_h}$. Let λ_0 (resp. λ_1) be the R-time for sink s_h with $\alpha_h = 0$ (resp. $\alpha_h = w_{b_h}$). In the process of identifying b_h, we would have done the λ_0-feasibility test and λ_1-feasibility test. Since the λ_1-feasibility test must have been successful, $\overline{\lambda}$ would have been set to λ_1 at that time. Moreover, $\overline{\lambda}$ could have been made smaller due to a later feasibility test. Thus changing λ with $\lambda \le \overline{\lambda}$ cannot move α_h beyond w_{b_h}. Similar argument shows that $\alpha_h > 0$ for $\lambda > \underline{\lambda}$. ◀

▶ **Corollary 18.** *Let $\underline{\lambda}$, $\overline{\lambda}$, and $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be as defined in Lemma 17. Then there is an optimal solution $\{\hat{s}_1, \ldots, \hat{s}_{k-1}; \hat{b}_1, \ldots, \hat{b}_{k-1}\}$ such that for $\forall h$ $(1 \le h \le i{-}1)$, $\hat{s}_h$ and s_h lie on the same vertex or same edge, and $\forall h$ $(1 \le h \le i{-}1) : \hat{b}_h = b_h$.*

In Fig. 7,[8] for $1 \le h \le i{-}1$, the value, $h\beta$, shown below b_h, indicates the amount by which α_h is reduced. Thus the difference between the values below b_h and b_{h+1} is β for all $1 \le h \le i{-}1$. To accommodate these changes in α_h and α_{h+1}, sink s_h must move to the left by the distance $\beta/2c\tau$ to balance its L-time and R-time. As a result, the evacuation time at each sink gets reduced to $\overline{\lambda}{-}\beta/2c$. Therefore, $(i{-}1)\beta$ is the increase in $\overline{\alpha}_{i-1}$ that must be sent to v. To make all the L-times and R-times the same, we should have $\lambda_v + (i{-}1)\beta/c = \overline{\lambda}{-}\beta/2c$, from which we get

$$\beta/2c = (\overline{\lambda}{-}\lambda_v)/(2i{-}1). \tag{17}$$

Then the L- and R-time of every sink equal

$$\overline{\lambda}_v = \overline{\lambda} - \beta/2c = \{(2(i{-}1)\overline{\lambda} + \lambda_v\}/(2i{-}1). \tag{18}$$

Clearly, this $\overline{\lambda}_v$ can be computed in $t(n) = O(1)$ time. We now perform the $\overline{\lambda}_v$-feasibility test to compare it with λ^*, which runs in $f(n) = O(k \log n)$ time by Lemma 16. If the $\overline{\lambda}_v$-feasibility test succeeds, then $\overline{\lambda}_v \ge \lambda^*$, which means that sink s_i needs to be placed at or to the left of v. If it fails, then $\overline{\lambda}_v < \lambda^*$, which means that sink s_i may be VB to v or it needs to be placed to the right of v. If the test is successful (resp. fails), we update $\overline{\lambda} \leftarrow \overline{\lambda}_v$ (resp. $\underline{\lambda} \leftarrow \overline{\lambda}_v$). This is repeated until we either locate a sink or reach vertex n.

The following lemma is the counterpart to Lemma 11.

▶ **Lemma 19.** *Suppose that the $\overline{\lambda}_v$-feasibility test failed, but the $\hat{\lambda}_{v+1}$-feasibility test succeeded. If the λ_v^+-feasibility test succeeds, where*

$$\lambda_v^+ = \{2(i{-}1)\overline{\lambda} + \lambda_{v+1}{-}l_v\tau\}/(2i{-}1), \tag{19}$$

then s_i is VB to v, otherwise, $s_i \in P(v, v{+}1]$.

[8] This is a special case of Fig. 8, and what is in the rest of this subsection follows from the analysis of the general edge capacity model in Sec. 5.1. But we present it here, since it is easier to understand the underlying idea with a simpler model.

Proof. λ_v^+ is obtained by replacing λ_v in (18) by $\lambda_{v+1} - l_v\tau$. Assume that the λ_v^+-feasibility test succeeds, i.e., $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) \geq \lambda^*$. In this case we could have either $s_i = v$ or $s_i = v^+$. But there is no advantage in placing it at v^+ over placing it at v, since they make no difference in the evacuation time of the overall solution.[9] If it fails, i.e., $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) < \lambda^*$, then we clearly have $s_i \succ v$.[10] ◀

As a result of the λ_v^+-feasibility test in Lemma 19, if s_i turns out to be VB to v, then we have identified a subpath of Type A or C. So we can isolate and discard the subpath ending at this sink s_i. But a copy of s_i should be made, because it is the start vertex of a subpath of Type C or D that comes next. If s_i is not VB, on the other hand, s_i lies on $(v, v{+}1)$, but we do not know exactly where: we only know that $d(v, s_i)$ depends linearly on λ, as implied by Lemma 17. We now proceed to determine the boundary vertex for s_i.

4.2 Finding next boundary vertex in optimal solution

We want to decide if boundary vertex b_i is a separator vertex, and once we have identified a separator vertex, we remove the edge incident to it from the right. Otherwise, we will only know that b_i will be split.

Assume that we have introduced i sinks so far in the current subpath, and as a result of the last successful $\overline{\lambda}$-feasibility test, the first $i{-}1$ sinks have the same L-time and R-time that are equal to $\overline{\lambda}$, which is the same as the L-time of s_i. We now reduce them by $\beta/2c$ from $\overline{\lambda}$ for some β to be determined below. Let λ'_b be the R-time at s_i for $P[s_i, b]$, where b is being tested as a possible boundary vertex for s_i. As we argued in Sec. 4.1, it should increase by $(2i{-}1)\beta/2c$, moving $\{s_h \mid 1 \leq h \leq i\}$ to the left by various distances. We thus have

$$\lambda'_b + (2i{-}1)\beta/2c = \overline{\lambda} - \beta/2c \Rightarrow \beta/2c = (\overline{\lambda}{-}\lambda'_b)/2i. \tag{20}$$

We now run a $\overline{\lambda}_b$-feasibility test for

$$\overline{\lambda}_b \triangleq \overline{\lambda} - \beta/2c = \{(2i{-}1)\overline{\lambda} + \lambda'_b\}/2i. \tag{21}$$

This is analogous to how we identifed VB sinks in Sec. 4.1. Here is the counterpart to Lemma 19.

▶ **Lemma 20.** *Suppose that the $\hat{\lambda}_{b-1}$-feasibility test failed, but the $\overline{\lambda}_b$-feasibility test succeeded. If the λ_b^--feasibility test succeeds, where*

$$\lambda_b^- = \{(2i{-}1)\overline{\lambda} + w_{b-1}/c\}/2i, \tag{22}$$

then $b{-}1$ is the separator vertex for the current subpath. If it fails, $b_i = b$ and b is a split vertex.

Intuitively, if the λ_b^--feasibility test succeeds, then $b{-}1$ cannot accept any more evacuees within the equalized time $\overline{\lambda}{-}\beta/2c$. If $b{-}1$ is the separator vertex, we end up with Type B or D. If $b_i = b$ is a split vertex, we do not compute its split portion α_i at this time. The updated $\overline{\lambda}$ is used to find the next sink, using it in Eq. (17).

[9] Moreover, there is no physical point corresponding to v^+ other than v.

[10] Note that if the critical vertex for $P[b_{i-1}, v]$ w.r.t. v is the same as that w.r.t. v^+, then $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) = \tilde{\Theta}_L^{[b_{i-1},\cdot]}(v) < \lambda^*$, so $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) \geq \lambda^*$ cannot happen.

▶ **Lemma 21.** *The total time to find all the λ values to be tested per sink and boundary vertex is $t(n) = O(k \log n)$, where $t(n)$ is defined in Lemma 5.*

Proof. We need to probe for the candidate vertex v for the next sink s_i (see Sec. 4.1) and boundary vertex b_i, and compute λ_v, $\overline{\lambda}_v$, and λ_v^+ (resp. λ'_b, $\overline{\lambda}_b$, and λ_b^-). The dominant cost is incurred for finding λ_v and λ'_b, which is $O(\log n)$ time, hence $t(n) = O(\log n)$. ◀

4.3 Time complexity of the algorithm

The dominant component of $h(n)$ in Lemma 5 is the time used to construct the CV tree $\mathcal{T}$, which is $O(n)$ by Lemma 7. Lemmas 5, 15, 16, and 21 now imply

▶ **Theorem 22.** *We can find the k-sink in path networks with uniform edge capacities in $O(n+k^2 \log^2 n)$ time.*

5 General edge capacity model

We want to make Lemma 17 and Corollary 18 valid for the general edge capacity model, but different edge capacities introduce some complications. First of all, we need to use the CUE tree $\mathcal{T}$, instead of the CV tree, to find various L-time and R-time. The main issue is computing the next λ value to be tested in optimization, and finding the time $t(n)$ needed to carry out this operation.

Assume that the sinks and boundary vertices have been placed up to boundary vertex b, and each sink has L-time and R-time equal to $\overline{\lambda}$. To locate the next sink, we want to identify two adjacent vertices v and $v{+}1$ such that $\tilde{\Theta}_L^{[b,\cdot]}(v)$ (resp. $\tilde{\Theta}_L^{[b,\cdot]}(v{+}1)$) is infeasible (resp. feasible). Similar operations need to be performed to find the next boundary vertex.

5.1 Evacuation time equalization

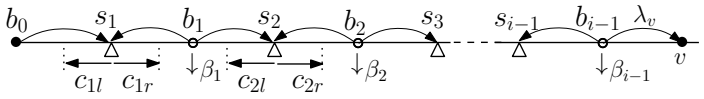

Figure 8 Evacuation time equalization.

5.1.1 Assuming the L- and R-critical vertices of each sink is unique

Fig. 8 corresponds to Fig. 7 in the uniform capacity model. Thus for $1 \le h \le i{-}1$, β_h is the reduction amount in α_h, and c_{hl} (resp. c_{hr}) is the capacity between sink s_h and its L-critical (resp. R-critical) vertex $\rho_L^{[b_{h-1},\cdot]}(s_h)$ (resp. $\rho_R^{[\cdot,b_h]}(s_h)$). If we start with the evacuation time based on the up-to-date $\overline{\lambda}$ and spread the slack to compute the equalized time λ for the feasibility test, then this λ value may not be the correct evacuation time at v, since λ was computed based on the (possibly wrong) assumption that $\rho_L^{[b_{h-1},\cdot]}(s_h)$ and $\rho_R^{[\cdot,b_h]}(s_h)$ would not change when λ is reduced from $\overline{\lambda}$. But in general, they may change. So the problem is how to ensure that critical vertices do not change. Our approach is to make the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently small so that as long as λ is varied within the constraint $\underline{\lambda} < \lambda < \overline{\lambda}$, they do not change We address this issue after presenting an evacuation time equalization method, assuming that the operations involved do not change critical vertices.

This assumption implies that capacities c_{hl} and c_{hr} do not change, as we move s_h to the left to equalize its L-time and R-time and reduce α_h by β_h, within the constraint that the value λ to be tested for feasibility satisfies $\underline{\lambda} < \lambda < \overline{\lambda}$.

Starting from the left end of the subpath of Type A that we are processing, if we reduce α_1 by β_1, the L-time and R-time of s_1 get reduced from $\overline{\lambda}$ to $\overline{\lambda}-\beta_1/2c_{1r}$. In general, reducing α_{h-1} by β_{h-1} (increasing $\overline{\alpha}_{h-1}$ by β_{h-1}) causes s_h to move to the left by some distance δ_h. Similarly, the decrease in the R-time at s_h is $\beta_h/c_{hr} - \delta_h\tau$. Equating these decreases, we get

$$\delta_h\tau - \beta_{h-1}/c_{hl} = \beta_h/c_{hr} - \delta_h\tau \Rightarrow \ \delta_h\tau = (\beta_{h-1}/c_{hl} + \beta_h/c_{hr})/2,$$

where β_0=0. We thus obtain

$$\delta_h\tau - \beta_{h-1}/c_{hl} = \beta_h/c_{hr} - \delta_h\tau = (\beta_h/c_{hr} - \beta_{h-1}/c_{hl})/2, \tag{23}$$

and the L-time and R-time get reduced from $\overline{\lambda}$ to

$$\overline{\lambda}-(\beta_h/c_{hr}-\beta_{h-1}/c_{hl})/2. \tag{24}$$

Equating the evacuation time reduction (23) for h=1 and h=2, we get

$$\beta_1/2c_{1r} = (\beta_2/c_{2r} - \beta_1/c_{2l})/2,$$

from which we can express β_2 as $\beta_2 = a_2\beta_1$ for constant $a_2 = c_{2r}(1/c_{2l} + 1/c_{1r})$. Equating the evacuation time reduction (23) for h=1 and h=3, we get

$$\beta_1/2c_{1r} = (\beta_3/c_{3r} - \beta_2/c_{3l})/2.$$

Substituting $\beta_2 = a_2\beta_1$ in this equality, we can express $\beta_3 = a_3\beta_1$ for some constant a_3. In general, we have $\beta_h = a_h\beta_1$ for a constant a_h, which is a function of capacities $\{c_{jl}, c_{jr} \mid 1 \le j \le h\}$. Moreover, a_h can be obtained in $O(1)$ time when h is incremented, since c_{jl} and c_{jr} ($j \le h-1$) remain the same by our assumption, hence it is a linear function of $\overline{\lambda}$ whose coefficients are known.

We can now determine β_1 by equating the L-time at v and the L-time at s_1 as follows.

$$\lambda_v + \beta_{i-1}/c(b_{i-1}, v) = \overline{\lambda} - \beta_1/2c_{r1}. \tag{25}$$

With this β_1, the L-time and R-time of every sink equal

$$\overline{\lambda}_v = \overline{\lambda} - \beta_1/2c_{r1}. \tag{26}$$

This equalized time is used for $\overline{\lambda}_v$-feasibility testing. The other types of subpaths can be analyzed similarly.

5.1.2 Making the L- and R-critical vertices of each sink unique

As part of preprocessing, for each node u of CUE tree $\mathcal{T}$, we construct array $\Lambda^w_R(u)$ from $A^w_R(u)$ by replacing each element W_h in it by the corresponding time, $\lambda^h_u \triangleq \Theta^w_R(W_h, u)$, which is the R-time w.r.t. $l(u)^-$ for $V(u)$ plus W_h coming from the right side of $r(u)$. In other words, (W_h, λ^h_u) is an inflection point. We define array $\Lambda^w_L(u)$ symmetrically to $\Lambda^w_R(u)$.

To make our assumption about the uniqueness of c_{hl} and c_{hr} valid, we update $\underline{\lambda}$ and $\overline{\lambda}$ on additional occasions, which makes the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently narrow. Consider, for example, a subpath of Type A and, without loss of generality, let $b_0 = 1$ be its leftmost vertex. In finding the edge on which s_1 should lie, using binary search, we look for two

adjacent vertices v and $v{+}1$ such that the L-time $\Theta_L^{[1,\cdot]}(v)$ (resp. $\Theta_L^{[1,\cdot]}(v{+}1)$) is infeasible (resp. feasible). By Lemma 9(b), we can compute $\Theta_L^{[1,\cdot]}(v)$ in $O(\log^2 n)$ time, and by the definition of v, we have

$$\Theta_L^{[1,\cdot]}(v) < \lambda^* \leq \Theta_L^{[1,\cdot]}(v{+}1). \tag{27}$$

So we can update the bounds by $\underline{\lambda} \leftarrow \max\{\underline{\lambda}, \Theta_L^{[1,\cdot]}(v)\}$ and $\overline{\lambda} \leftarrow \min\{\overline{\lambda}, \Theta_L^{[1,\cdot]}(v{+}1)\}$. Thus the optimal s_1^* will lie on $P[v, v{+}1]$. Ignoring the special case of $s_1^* = v{+}1$, let $s_1^* \in e_v = (v, v{+}1)$. Then it is clear that the L-critical vertex w.r.t. s_1^* is the same as the L-critical vertex w.r.t. $v{+}1$, and we already know it is unique at this point.

Next, we look for the boundary vertex b_1 for s_1. Let $\overline{s}_1$ be the position of the first sink, whose L-time is $\overline{\lambda}$ that is the most up-to-date upper bound. Using binary search, we probe vertex $b \succ \overline{s}_1$, computing R-time $\lambda_b = \Theta_R^{[\cdot,b]}(\overline{s}_1) < \overline{\lambda}$.[11] Using this λ_b, we equalize the L-time and R-time of s_1, and test the equalized value, $\overline{\lambda}_b$, for feasibility. This way, after one feasibility test per probed vertex, we can identify two adjacent vertices b and $b{+}1$ such that $\Theta_R^{[\cdot,b]}(s_1)$ is infeasible and $\Theta_R^{[\cdot,b+1]}(s_1)$ is feasible, where s_1 is the sink with the equalized L-time and R-time. Note that these tests are counted in $cmp(n)$.

Before proceeding further, we want to make sure that any $s_1 \in (v, v{+}1)$ considered in the future will have a unique R-critical vertex, for $P[s_1, b{+}1]$, as long as $\lambda \in (\underline{\lambda}, \overline{\lambda})$ for the most up-to-date $\underline{\lambda}$ and $\overline{\lambda}$ and that $\alpha_{b+1} > 0$ holds. To this end, we need to narrow the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently. More generally, let $\underline{\lambda}$ and $\overline{\lambda}$ be the current bounds, and let $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be the non-VB sinks and split boundary vertices, based on the current $\overline{\lambda}$ that resulted from the last successful feasibility test. Suppose that $s_{i-1} \in (v, v{+}1)$ and $b_{i-1} = b{+}1$.

For each $u \in N[v{+}1, b]$, we do binary search in $\Lambda_R^w(u)$, and based on the probed value, we first equalize the L-time and R-time of $\{s_h \mid 1 \leq h \leq i{-}1\}$. We then perform $O(\log n)$ feasibility tests to identify two adjacent values, $\lambda_u^{g_u}, \lambda_u^{g_u+1} \in \Lambda_R^w(u)$ such that the feasibility test for the corresponding equalized value, $\overline{\lambda}_u^{g_u}$ (resp. $\overline{\lambda}_u^{g_u+1}$) fails (resp. succeeds). We now update the bounds by

$$\underline{\lambda} \leftarrow \max\left\{\underline{\lambda}, \max_{u \in N[v+1,b]}\{\overline{\lambda}_u^{g_u}\}\right\}, \tag{28}$$

$$\overline{\lambda} \leftarrow \min\left\{\overline{\lambda}, \min_{u \in N[v+1,b]}\{\overline{\lambda}_u^{g_u+1}\}\right\}. \tag{29}$$

What we have done so far essentially is to identify a unique critical vertex per node $u \in N[v{+}1, b]$. We need $O(\log n)$ more feasibility tests to make the critical vertex for $P[v{+}1, b]$ w.r.t. $s_i \in (v, v{+}1)$ unique.

▶ **Lemma 23.** *Let $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be the non-VB sinks and split boundary vertices, based on the current $\overline{\lambda}$ that resulted from the last successful feasibility test. For any equalized value $\lambda \in (\underline{\lambda}, \overline{\lambda})$, where $\underline{\lambda}$ (resp. $\overline{\lambda}$) is given by* (28) *(resp.* (29)*), the R-critical vertex for any $s_{i-1} \in (v, v{+}1)$ is unique, and we can compute* (28) *and* (29) *in $O(f(n)\log^2 n)$ time.*

Proof. For any $u \in N[v{+}1, b]$, we have $\underline{\lambda} \geq \overline{\lambda}_u^{g_u}$ and $\overline{\lambda} \leq \overline{\lambda}_u^{g_u+1}$. Therefore, $\lambda \in (\underline{\lambda}, \overline{\lambda})$ implies that λ lies in a unique position among the values in $\Lambda_R^w(u)$.

For each of the $O(\log n)$ probed values from $\Lambda_R^w(u)$, we can evaluate the equalized value $\overline{\lambda}_u^{g_u}$ in $O(k)$ time, and $\overline{\lambda}_u^{g_u}$ can be tested for feasibility in $f(n)$ time. This is repeated $O(\log n)$ times, resulting in $O(f(n)\log n)$ time. The total time for all nodes u in $N[v{+}1, b]$ is thus $O(f(n)\log^2 n)$. ◀

[11] Vertex b with $\Theta_R^{[\cdot,b]}(\overline{s}_1) \geq \overline{\lambda}$ is of no interest if $s_1^* \prec \overline{s}_1$.

This ensures that if any λ value satisfying $\underline{\lambda} < \lambda < \overline{\lambda}$ is tested for feasibility in the future, the unique critical vertex $\rho_R^{[\cdot,b+1]}(s_{i-1})$ is already known. Since we assume a subpath of Type A, $b{+}1$ is split in the optimal solution. Moreover, we can easily find the bounds on α_{b+1} corresponding to $\underline{\lambda}$ and $\overline{\lambda}$.

Let b_{i-1} be the last boundary vertex introduced, which is split, and consider vertex $v \succ b_{i-1}$ which is being probed as a candidate for s_i. See v in Fig. 8. We want to narrow the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently, so that the L-critical vertex for the subpath $P[b_{i-1}, s_i]$ w.r.t. s_i is unique when λ-feasibility is tested for any $\lambda \in (\underline{\lambda}, \overline{\lambda}]$. To this end, we do binary search in $\Lambda_L^w(u)$ for each $u \in N[b_{i-1}{+}1, v]$. Let $\lambda_u^{g_u}$ be the probed element. Using $\lambda_u^{g_u}$, we equalize the L-time and R-time of all s_h $(1 \le h \le i{-}1)$, and perform the feasibility test for the resulting equalized value $\overline{\lambda}_u^{g_u}$. This way we can identify two adjacent values $\lambda_u^{g_u}$ and $\lambda_u^{g_u+1}$ in $\Lambda_L^w(u)$ such that the feasibility test for the equalized value $\overline{\lambda}_u^{g_u}$ (resp. $\overline{\lambda}_u^{g_u+1}$) fails (resp. succeeds). The rest is similar to what we did for sink s_{i-1} above. This ensures that if any λ value satisfying $\underline{\lambda} < \lambda < \overline{\lambda}$ is tested for feasibility in the future, the L-critical vertex for the subpath $P[b_{i-1}, s_i]$ w.r.t. s_i is unique and already known. To summarize,

▶ **Lemma 24.** *Let $\{s_1, \ldots, s_i; b_1, \ldots, b_{i-1}\}$, where $s_i \in (v, v{+}1]$, be the sinks and split boundary vertices that resulted from the last successful feasibility test. In $O(f(n) \log^2 n)$ time, we can further reduce the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently, so that if any value $\lambda \in (\underline{\lambda}, \overline{\lambda})$ is tested for feasibility, the L-critical vertex for any s_i is unique.*

Clearly, the uniqueness of the critical vertices implies the uniqueness of the capacities $\{c_{hl}, c_{hr} \mid 1 \le h \le i{-}1\}$ in Fig. 8. Note that if the equalized λ lies outside the current interval $(\underline{\lambda}, \overline{\lambda})$, then we immediately know if it is feasible or not.

5.2 Complexity

▶ **Lemma 25.** *If the CUE tree $\mathcal{T}$ is available, then for any λ, we can decide λ-feasibility in $f(n) = O(k \log^2 n)$ time.*

Proof. The proof is similar to that of Lemma 16, except that it takes $O(\log^3 n)$ time to identify the maximal λ-interval by binary search, using the comment after Lemma 9. ◀

Since we already know $h(n)$, $cmp(n)$, and $f(n)$, the only remaining task is to find $t(n)$. Lemmas 23 and 24 imply

▶ **Lemma 26.** *If the CUE tree $\mathcal{T}$ is available, generating the values to be tested for feasibility takes $t(n) = O(f(n) \log^2 n) = O(k \log^4 n)$ time.*

The CUE tree can be constructed in $h(n) = O(n \log n)$ time by Lemma 7. We have $cmp(n) = O(k \log n)$, just as for the uniform capacity model. Finally, Lemmas 5, 25, and 26 imply our second main theorem.

▶ **Theorem 27.** *For the general edge capacity model, we can find the optimal k-sink in $O(n \log n{+}k^2 \log^5 n)$ time*

6 Cycle networks

We shall show that the feasibility test for non-confluent flows on cycle networks can be performed in time that is no more than n times the time needed by the feasibility test for path networks. It is known that [3, 7] the k-sink problem for cycle networks with confluent flows can be solved in $O(n \log n)$ (resp. $O(n \log^3 n)$) time when the edge capacities are uniform (resp. general). Given a cycle network $C = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the the vertex set, and the edge set E is given by $E = \{(i, i{+}1) \mid i = 1, 2, \ldots, i{-}1\} \cup \{(n, 1)\}$.

▶ **Lemma 28.** *There is always an optimal solution in which either at least one sink is located at a vertex or there is an edge that carries no flow.*

Proof. To prove the above claim, for a cycle C, let $s_1^*, s_2^*, \ldots, s_k^*$ be the locations of optimal sinks arranged clockwise, and b_i be the boundary vertices between s_i^* and s_{i+1}^*. If a sink is on a vertex, then we can divide it into two vertices, converting C into a path. If the boundary vertex b_i for sink s_i^* is not split, then cutting the edge $(b_i, b_i{+}1)$ converts C into a path.

Therefore, assume that all the optimal sinks are on edges and that all boundary vertices are split. Pick a sink s_i^*, and gradually move it clockwise. We adjust the split portions of the boundary vertices to compensate for this move. Eventually one of the two things will occur. (i) some sink s_j^* reaches a vertex,[12] or (ii) a boundary vertex b_l is no longer split. In the former case, we cut the vertex into two vertices, which become the end vertices of the resulting path. In the latter case, the edge between b_l and $b_l{+}1$ will carry no flow any more, and it can be removed, resulting in a path from $b_l{+}1$ to b_l. ◀

This implies that we can solve the problem as follows. We create n paths by dividing each vertex into two vertices, and another n paths by removing each edge. We then solve the k-sink problem for each of these $2n$ paths. The solution with the minimum evacuation time is our overall solution. Let P_i denote the path that results by removing edge $(i, i{+}1)$, where vertex $n{+}1$ is interpreted as 1. We can solve the problem for each P_i in $O(n \log n{+}k^2 \log^5 n)$ time by Theorem 27. Thus the total time for all such paths is given by

$$O(n^2 \log n{+}k^2 n \log^5 n). \tag{30}$$

However, we can save time on the preprocessing time $h(n)$ as follows. We make a copy of path P_n, name its vertices $\{n{+}1, n{+}2, \ldots, 2n\}$, and connect P_n and its copy by introducing a new edge $(n, n{+}1)$. This results in a path P' of length $2n{-}1$. We now construct the CUE tree for this P'. In solving the problem for P_i for any i, whenever the value $\Theta_L^{[b_{i-1},\cdot]}(v)$ or $\Theta_R^{[\cdot,b]}(s_i)$ is needed, we can use a portion of the CUE tree to compute it. Since we can construct this CUE tree in $O(n \log n)$ time, we have $h(n) = O(n \log n)$. This implies that we can replace the first term in (30) by $n \log n$, which is dominated by the second term.

▶ **Theorem 29.** *If the edge capacities are uniform (resp. general), we can find the optimal k-sink in cycle networks in $O(k^2 n \log^2 n)$ (resp. $O(k^2 n \log^5 n)$) time.*

7 Conclusion and discussion

We have presented algorithms to find a k-sink on path networks when the flow is non-confluent. For the uniform capacity model, the time complexity of our algorithm is asymptotically the same as the corresponding algorithm for confluent flow discussed in [4]. For the general capacity model, however, it takes longer than the corresponding algorithm in [4]. We showed that a similar approach can be used to find a k-sink on cycle networks, but the time complexity increases.

A model in which the sinks are constrained to be in a prescribed set of vertices might be more realistic. We can apply our methods developed in this paper with only small changes to find a solution in such a model.

[12] $j{=}i$ is possible.

There are a few open problems. First of all, can we improve the feasibility test for cycle networks? Tree networks appear much harder to deal with. For the confluent flow model, Chen and Golin [6] propose an $O((k+\log n)k^2 n \log^3 n)$ (resp. $O((k+\log n)k^2 n \log^4 n)$) time algorithm for finding a k-sink in the uniform (resp. general) capacity model. One of the difficulties in the non-confluent flow model for tree networks is that a split portion of a vertex cannot be represented by just one variable α_i per vertex, because a vertex may have many neighbors. Another serious problem is that the optimal split portion may be time-dependent.

References

1. P.K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30(4):412–458, 1998.
2. G.P. Arumugam, J. Augustine, M. Golin, Y. Higashikawa, N. Katoh, and P. Srikanthan. Optimal evacuation flows on dynamic paths with general edge capacities. *CoRR abs/1606.07208*, pages 1–37, 2016.
3. R. Benkoczi and R. Das. The min-max sink location problem on dynamic cycle networks. Unpublished manuscript, October 2018.
4. B. Bhattacharya, M. Golin, Y. Higashikawa, T. Kameda, and N. Katoh. Improved algorithms for computing k-sink on dynamic flow path networks. In *Proc. Algorithms and Data Structures Symp., Springer-Verlag, LNCS 10389*, pages 133–144, 2017.
5. B. Bhattacharya and T. Kameda. Improved algorithms for computing minmax regret sinks on path and tree networks. *Theoretical Computer Science*, 607:411–425, November 2015.
6. D. Chen and M.J. Golin. Minmax centered k-partitioning of trees and applications to sink evacuation with dynamic confluent flows. In *CoRR abs/1803.09289*, 2018.
7. R. Das. Minmax sink location problem on dynamic cycle networks. Master's thesis, University, of Lethbridge, Lethbridge, Canada, 2018.
8. L.R. Ford and A.D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.
9. M.J. Golin, H. Khodabande, and B. Qin. Non-approximability and polylogarithmic approximations of the single-sink unsplittable and confluent dynamic flow problems. In *Proc. 28th Int'l Symp. on Algorithms and Computation (ISAAC)*, pages 41:1–41:13, 2017.
10. 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.
11. Y. Higashikawa, M.J. Golin, and N. Katoh. Multiple sink location problems in dynamic path networks. *Theoretical Computer Science*, 607(1):2–15, 2015.
12. Y. Higashikawa and N. Katoh. A survey on facility location problems in dynamic networks. *The Review of Socionetwork Strategies*, 13:163–208, September 2019.
13. B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000.
14. N. Kamiyama. *Studies on Quickest Flow Problems in Dynamic Networks and Arborescence Problems in Directed Graphs: A Theoretical Approach to Evacuation Planning in Urban Areas.* PhD thesis, Kyoto University, 2005.
15. S. Mamada, K. Makino, and S. Fujishige. Optimal sink location problem for dynamic flows in a tree network. *IEICE Trans. Fundamentals*, E85-A:1020–1025, 2002.
16. N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.
17. M. Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization, W. Cook, L. Lovasz, and J.Vygen, Eds.*, pages 451–482. Springer Verlag, 2009.