MINMAX SINK LOCATION PROBLEM ON DYNAMIC CYCLE NETWORKS

RAJIB CHANDRA DAS
Bachelor of Science, Chittagong University of Engineering and Technology, 2014

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

MINMAX SINK LOCATION PROBLEM ON DYNAMIC CYCLE NETWORKS

RAJIB CHANDRA DAS

Date of Defence: December 12, 2018

Dr. Robert Benkoczi Supervisor	Associate Professor	Ph.D.
Dr. Shahadat Hossain Committee Member	Professor	Ph.D.
Dr. Saurya Das Committee Member	Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To my mom who sacrificed her whole life to build up mine.

Abstract

We address both 1 and k sink location problems on dynamic cycle networks. Our 1-sink algorithms run in $O(n)$ and $O(n\log n)$ time for uniform and general edge capacity cases, respectively. We improve the previously best known $O(n\log n)$ time algorithm for single sink introduced by Xu et al. [Xu et al. 2015] with uniform capacities. When k¿1, we improve two results [Benkoczi et al. 2017] for both with uniform and arbitrary capacities by a factor of $O(\log n)$. Using the same sorted matrices optimization framework originally devised by Frederickson and Johnson and employed by [Benkoczi et al. 2017], our algorithms for the k-sink problems have time complexities of $O(n\log n)$ for uniform, and $O(n\log^3 n)$ for arbitrary capacities. Key to our results is a novel data structure called a cluster head forest, which allows one to compute batches of queries for evacuation time efficiently.

Acknowledgments

I am very much grateful to my thesis supervisor, Dr. Robert Benkoczi for his unconditional boundless support. I am blessed that most of the times I received his insightful suggestions and inspirations to concentrate more on my research. I want to thank him for everything he did for me. I would like to thank the other members of my thesis committee, Dr. Shahadat Hossain and Dr. Saurya Das for spending their valuable time to read my thesis proposal and the final thesis. I would also like to thank all of my office colleagues. I am thankful to Polash, who cooked me food while I was writing my research paper and this thesis.

Last but not least, I am very much indebted to my family members, I can never repay them for the support they have provided throughout my studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Facility location problem (FLP) asks for the location of facilities so that we can meet the specific objectives such as minimizing transportation cost and the distance between customer and facilities etc. Besides transportation cost, we have to consider the cost of placing facilities. There are different kinds of FLP. The minmax problem, for instance, explores the placement of the facility with the objective to minimize the maximum distance from a customer to the nearest facility.

There is another type of location problem known as sink location problem (SLP). Fire escape arrangement, earthquake or tsunami evacuation plan are the classical applications of this problem. Unlike the FLP in sink location problem we have to take in account new constraints, capacity (uniform or arbitrary) on every edge denoting the number of supplies can enter an edge in one unit time and transit time per unit distance. If all the edge capacities are equal, then the problem is called the uniform otherwise arbitrary or general edge capacity problem. If the capacity of an edge is comparatively smaller than the vertex supplies, then the supplies take more time to cover the edge. In [14] Hamacher and Tjandra showed that the sink location problem could be represented by a network whose vertices represent the initial location of supplies (evacuees) and the edges represent the possible evacuation routes. This network is called the *dynamic graph network*, which was first introduced by Ford and Fulkerson [10] in 1958.

Dynamic networks can be represented in *discrete* and *continuous model*. When all the input values are given as an integer (resp. as a real number) then the model is called the discrete (resp. continuous) model. In [3], Baumann, and Skutella showed that dynamic graph networks could be used for evacuations model. In the evacuation problem [14, 18], where the weights (i.e., evacuees) of each vertex are discrete and the sinks has infinite demands. We can represent the total number of people in a building as the supplies (evacuees) of a vertex in the network. The edges can represent the roads, and the sinks are the evacuation buildings as described in [1]. The outcome is to determine the optimal locations for evacuation buildings so that everyone can reach a sink as best as possible by minimizing the evacuation time.

In [17], Kariv, and Hakimi proved that for a general graph the $k-$center problem is NP-hard. In the center problem, to compute the evacuation time we only need to determine the maximum distance from the vertices to the nearest facilities. We notice that in our minmax sink location problem if the edge capacities are too large, then it can be reduced to the $k-$center problem. So, we can also prove that the general $k-$sink problem is NP-hard. To solve the $k-$sink location problem for restricted topology networks (e.g., dynamic paths and trees), the efficient way is to find the vertices for which the supplies of other vertices get congested. If the supply of a vertex requires more time to move to the sink because of the small edge capacity than the supplies from other vertices, then the congestion occurs. When the edge capacities are the same for all edges, it is easy to find out the exact location where congestion occurs. This problem is known as sink location problem with uniform edge capacity. On the contrary, the sink location problem with different edge capacities (or arbitrary capacities) is way more difficult.

In this thesis, we study the sink location problem on dynamic cycle networks for both uniform and arbitrary capacities. A *simple cycle* is a trail of edges and vertices if and only if starting vertex and ending vertex is identical and other edges or vertices should not be traversed more than once. We improve the time complexity of existing algorithm for both the 1 and k sink location problems in cycles. For a cycle, given a sink and a vertex, there are two paths (clockwise and counter-clockwise) the supply can reach the sink. Here, the big challenge is to find the right path for the supply from a vertex travels towards a sink. To solve the sink location problem on the dynamic in other simple networks (path and tree) the common complication (details in the following section) is finding the vertex for which the congestion occurs. In the cycle network, this congestion can also happen. Specifically, we need to know as fast as possible about the congestion in a cycle network. In this thesis, we assume the flow of the supply is *confluent*, that means every vertex has maximum one outgoing edge. As a result, we can observe that on a cycle network with 1-sink, there exists an edge which will not carry flow, which we call a *split edge.*

Definition 1.1. (Split edge). The edge not traversed by any flow in a single sink location problem on a cycle called the split edge.

After splitting an edge, we can solve the path problem by the most efficient existing algorithm. However, the big question is to find out a particular splitting edge in an efficient way for which the evacuation time is optimal. We call this specific edge an *optimal splitting edge*.

Definition 1.2. (Optimal Splitting Edge). A specific splitting edge for which the evacuation completion time is minimum is called the optimal splitting edge.

Therefore, throughout this thesis, we explore how to efficiently find the optimal splitting edge and solve the sink location problem on cycle network for a discrete model with no more than the time complexity of the existing path algorithm.

1.1 Literature Review

In the literature, there are a handful of interesting articles on sink location problems have been published for different dynamic networks with both uniform and general edge capacities. Most of the previous results are exists for the special graphs such as paths and trees which are graphs without cycles. Very few articles are available for placing sink on cycle networks. Table 1.1 represents the survey results on previous research.

Benkoczi et al. [4] solved the $p-$center problem on cycle networks that runs in $O(n \log n)$ time, where n is the number of vertices on the given cycle. The authors also studied the multiple sink location problems on dynamic cycle networks and designed an $O(n \log^2 n)$ and $O(n \log^4 n)$ time algorithms for finding the optimal $k-$sinks in dynamic flow cycle networks with uniform and general edge capacities, respectively. For any given cycle network, by splitting an arbitrary edge of the cycle, the authors generated a path. They doubled the path by combining two copies of the obtained path. To solve the multiple center or sink location problems the authors perform the feasibility test which decides for any given number of facilities whether the supply of every vertex can be served by a facility or not within a given time. Regarding the sink location problem, the authors introduced *Capacity and Upper Envelopes tree* (CUE tree), whose leaves are the vertices of the path. At every node of the CUE tree, the leftmost vertex of that node, the rightmost vertex of that node, two weight functions, and two capacity functions was stored. To compute the evacuation time for a given path, one needs to know about the congestion. In this regard, the CUE tree provides all the necessary information to determine the congestion more efficiently.

Higashikawa et al. [16] studied the multiple sinks ($k-$sink) location problems in a dynamic path network when the edge capacities are uniform. The authors computed the optimal cost for two criteria, the minimum of maximum cost and minimum of total cost. To calculate the optimal cost, at first the authors illustrated their model for solving the 1-sink location problem and then extended it to the $k-$sinks problem. They have formulated a recursive function to solve the main problem: locating $k-$sinks in a path. At every subproblem, they have used $1-$sink algorithm. The authors used a $(k-1)$-dimensional vector denoting the $(k-1)$ divider which divides all supplies (evacuees) between two consecutive sinks into two separate groups. The supplies to the left (resp. right) of the divider evacuate to the left (resp. right) sink. The authors solved the multiple sink location problems on dynamic path networks in $O(kn)$ time.

Table 1.1: Tabular representation of previous results

Net-works	No. of Sink	Edge Capacity	Running Time	Ref
Tree	$k = 1$	uniform	$O(n \log n)$	[19]
		arbitrary	$O(n \log^2 n)$	[19]
	$k > 1$	uniform	$O(max\{k, \log n\} kn \log^3 n)$	[8]
		arbitrary	$O(max\{k, \log n\} kn \log^4 n)$	[8]
Path	$k = 1$	uniform	$O(n)$	[16]
		arbitrary	$O(n \log n)$	[1]
	$k > 1$	uniform	$O(kn)$	[16]
			• $O(n + k^2 \log^2 n)$ when k is $o(\sqrt{\frac{n}{\log n}})$ • $O(n \log n)$ when k is $\Omega(\sqrt{\frac{n}{\log n}})$	[7]
		arbitrary	$O(nk \log^2 n)$	[1]
			• $O(n \log n + k^2 \log^4 n)$ when k is $o(\sqrt{\frac{n}{\log n}})$ • $O(n \log^3 n)$ when k is $\Omega(\sqrt{\frac{n}{\log n}})$	[7]
Cycle	$k > 1$	uniform	$O(n \log^2 n)$	[4]
		arbitrary	$O(n \log^4 n)$	[4]

Bhattacharya et al. [7] improved the algorithms for placing multiple sinks on dynamic flow path networks. The authors studied both uniform and general edge capacity cases. The authors stored all necessary information to determine the congestion in the CUE tree and extracted the information efficiently in demand. In this paper, multiple sink location problems have been solved by testing the feasibility. To optimize their proposed algorithm the authors used two different frameworks (details in chapter 5) and obtained $O(n \log n)$ and $O(n + k^2 \log^2 n)$ time algorithms when all edges have the same capacity. For general edge capacities, the authors presented two different results that run in $O(n \log^3 n)$ and $O(n \log n + k^2 \log^4 n)$.

1.2 Contribution of this thesis

In this thesis, our main contribution is to propose a novel data structure which we call *Cluster Head Forest* (CHF for short). In the preprocessing phase, we construct the CHF with the necessary information about congestion. The CHF allows us to answer a query for computing the evacuation time for any given subpath as well as a set of queries. The real power of the CHF comes to solve a batch of queries in an efficient manner. To solve the multiple sink location problems, we perform the feasibility test. Using our CHF, more efficiently we can test the feasibility.

Table 1.2: Summary of our contribution in this thesis

No. of sink	Optimization framework	Edge capacities	Running time
$k = 1$	NA	Uniform	$O(n)$
	NA	General	$O(n \log n)$
$k > 1$	Parametric Search	Uniform	$O(n + kn \log n)$
	Parametric Search	General	$O(n \log n + kn \log^2 n)$
	Sorted Matrix	Uniform	$O(n \log n)$
	Sorted Matrix	General	$O(n \log^3 n)$

Table 1.2 represents the summary of our results. For the first time, we propose a single sink location problem on cycle networks with general edge capacities. Besides this, we improve all known algorithms for dynamic cycle network that have been studied so far. We

solve 1 and k-sink location problems in cycle networks with the same time complexity as the currently best algorithms for path networks. The comparison with our results and the existing algorithms are as follows:

1. We show that $1-$sink location problem on cycle networks with uniform capacities can be solved in $O(n)$ time. Our result improves the algorithm developed by Xu and Li [21] when the edge capacities are same, whose time complexity is $O(n \log n)$.
2. We solve the multiple sink location problems on cycle networks with uniform edge capacities in $O(n \log n)$ time. We improve the very recent algorithm which is proposed by Benkoczi et al. [4]. Their algorithm can run in $O(n \log^2 n)$ time.
3. We provide an $O(n \log^3 n)$ time algorithm for locating a $k-$sink on cycle networks with arbitrary capacities, whereas the previously best-known algorithm as presented in [4] requires $O(n \log^4 n)$ time.

To obtain our claimed results, we use our proposed data structure that allows us to perform a faster feasibility test for finding the multiple sinks on the path. The above discussion summarizes that we improve all known algorithms by a factor of $\log n$ and propose a new algorithm for the single sink location problem with arbitrary edge capacities.

1.3 Organization of the thesis

We organized the thesis as follows.

In the following chapter, we define our model, discuss preliminary concepts and the terms that are used throughout the thesis. We also describe the procedure to generate a path from the given cycle by splitting an edge. Then we focus on how to determine the optimal splitting edge which helps us to find out the minimum evacuation time.

In chapter 3, we introduce our proposed novel data structure. We construct cluster head forests for both uniform and non-uniform edge capacities. How can our forests solve the sink location problems more efficiently is also discussed in this chapter.

Chapter 4 presents single and multiple sink location algorithms for both uniform and general edge capacities. When the edge capacities are arbitrary, we are the first who propose an algorithm for solving the single sink problem on cycle. When the number of sinks $k > 1$, our algorithm can run in the same complexity of the existing path algorithm. We also show that our algorithms improved some previous results.

In chapter 5, we provide several algorithms using two existing optimization frameworks. One is better than others based on the given value of k.

Finally, we conclude the thesis with the future research directions and summarizing our research.

Chapter 2

Preliminaries

2.1 Introduction

In this chapter, we define our model and essential terms that are used throughout the thesis. We also discuss the necessity of our proposed data structure to overcome the time complexity of the trivial algorithm. Section 2.2 comprises the definition of our proposed model, necessary notations and terms that will be used in the following chapters. In the next section, we include the elementary discussion about evacuation completion time. In section 1.1, we discuss a brute force algorithm for placing sink on a cycle network. This chapter concludes with providing an algorithm for the $1-$sink location on a cycle that leads us to design a new data structure to solve the algorithm more efficiently.

2.2 Model definition

Let $N = (C = (V,E), l, w, c, \tau)$ be our proposed dynamic network that comprises cycle C, which consists of vertices $V = \{v_1, v_2, v_3, \ldots, v_n\}$ and edges $E = \{e_1, e_2, \ldots, e_n\}$ where $e_i = (v_i, v_{i+1})$; we let l_i be the positive length of edge e_i, $l_i \in R^+$; w_i represents the weight (= supply) of vertex v_i with positive value, $w_i \in R^+$; c_i is the edge capacity of e_i (uniform or arbitrary capacity, defined in chapter 1), which is the maximum limit of supply that can enter an edge per unit time; we denote τ as the transit time per unit distance, $\tau \in R^+$.

As we assume the vertices are indexed from 1 to n, i.e., $v_1, v_2, \ldots, v_n$. So, we can find a vertex with its index. The set of vertices V denoted as $v_1 \leq v_2 \leq, \ldots, \leq v_n$. For any two points $(x, y) \in C$, we mean that these two points can be anywhere on the cycle

C. Let $V[x,y]$ denote the set of vertices between v_x and v_y. If $x \leq y$, then $V[x,y]$ consists of $v_x, v_{x+1}, \ldots, v_{y-1}, v_y$; otherwise, if $x > y$, $V[x,y] = v_y, v_{y+1}, \ldots, v_{n-1}, v_n, v_1, \ldots, v_{x-1}, v_x$. Similarly, by $P[x,y]$ we mean the subpath from x to y induced by $V[x,y]$. We let $d(x,y)$ to be the length from point x to y, when $x \leq y$. If $y \leq x$, then the distance from y to x is determined by $d(y,x)$. Thus, $d(x,y)$ and $d(y,x)$ are not necessarily be equal. The total order to the set of points P on the edges of the cycle can be represented as $P \in C$, in such a way that if point $a \in P[x,x+1]$ and $b \in P[y,y+1]$, then $a \leq b$ if $x+1 \leq y$.

Let a point x lies on edge $e_i \in E$. We mean the point x can be located anywhere from vertex v_i to v_{i+1} thus the sum of the distances from the vertices v_i and v_{i+1} to the point x is equals to the edge length l_i. In other words, we can define the distance from x to the endpoints so that the edge length is preserved. The distances from vertex v_i to the point x and v_{i+1} to x are $d(v_i,x)$ and $d(x,v_{i+1})$, respectively. So, mathematically we can write, $d(v_i,x) + d(x,v_{i+1}) = l_i$. Similarly, we let $c(x,y)$ stands for the minimum capacity from x to y. So, the capacity from v_i to x and from x to v_{i+1} is same as of e_i, which is c_i. For $i = 1,2,\ldots,n$, we let v_i^+ (resp. v_i^-) be a point on edge e_i (resp. e_{i-1}) which is just right (resp. left) of vertex v_i.

We let $W[v_i,v_j]$ be the sum of supplies (= weights) of vertices from $V[v_i,v_j]$. We calculate this weight array in preprocessing stage by spending linear time so that we can get the sum of weights in constant time for any given starting and ending vertex. If the number of vertices on the given cycle is n, then consider a path $P[v_1,v_{2n}]$ (i.e., $v_1,\ldots,v_n,v_1,\ldots,v_n$ where $v_{n+i} = v_i$). We can compute the array as follows [7]

$$W[i] = \sum_{v_j \in V[v_1,v_j]} w_j \quad \text{for } 1 \leq i \leq 2n \tag{2.1}$$

then for any arbitrary interval from v_i to v_j we can determine the sum of weights, $W[v_i,v_j] = W[j] - W[i-1]$, when $i \leq j$.

2.2.1 Flow and properties of sinks

In our proposed model, we assume a sink node holds all the properties that we described above for a point. That means a sink s cannot be on the vertices only, but it can be placed on the edges too, $s \in C$. We also assume that supply of a particular vertex cannot be split, i.e., all supplies of a vertex will evacuate to the same sink. If a sink is placed on vertex, then we call it the *sink vertex*. The supply of sink vertex will be evacuated instantly (in time 0) because the supplies do not need to move anywhere for evacuation. A sink is the safest place for all suppliers who want to access it. A sink has an infinite capacity to hold all supplies come to evacuate. The supplies coming from clockwise and anti-clockwise directions of a sink can easily reach without any interruption by each other. However, note that the supplies moving in the same direction can be disrupted, in details will be discussed in the following sub-section.

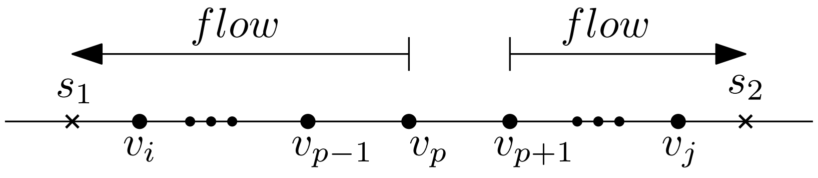

Figure 2.1: The flow of vertex v_p (resp. v_{p+1}) is towards the sink s_1 (resp. s_2)

We also assume that the flow of supply is confluent, that means the evacuees of all vertices start their journey simultaneously to a sink such that every vertex has maximum one outbound edge. Consider a subpath $P[v_l, v_r]$ with two sinks s_1 and s_2 as depicted in Fig. 2.1. For any cycle vertex $v_h \in V[v_i, v_j]$ the flow is confluent, meaning the supply of v_h will travel either towards s_1 or s_2. Let v_p evacuates at s_1, then the supply of vertices $V[v_p, v_i]$ must move in the counter clockwise direction for evacuating to the sink s_1. Similarly, if v_{p+1} moves to the sink s_2, the supplies of $V[v_{p+1,v_j}]$ also evacuate to s_2.

2.2.2 Sink location problem definition

Using the analysis from [6], let $\mathcal{F}(X)$ is the set of all confluent flows, where X is another set of $k-$sinks to be located on the given cycle and $k \leq n$. We denote by $\Theta(F)$ the evacuation time for a flow $F \in \mathcal{F}(X)$. We let $\Theta(X)$ be the evacuation time for all the sinks of a set $X, \Theta(X) = \min_{F \in \mathcal{F}(X)} \Theta(F)$. Then the $k-$sink location problem can be defined as

Problem 2.1. [6] Given a dynamic cycle network $N = (C, l, w, c, \tau)$, where $C = (V, E)$ is a cycle graph, and an integer $k \leq n$, find a set $X^* \subseteq N$ of points on the network which may include the edges of the network, so that $|X| = k$ and

$$\Theta(X^*) = \min_{X \subseteq N} \Theta(X), \text{ where } \Theta(X) = \min_{F \in \mathcal{F}(X)} \Theta(F).$$

2.2.3 Congestion and evacuation time

We need to determine the evacuation time for $P[v_p, v_n]$, where $1 \leq p \leq n$. Given the evacuation time of $P[v_, v_n]$ with sink on v_i^-, we can obtain the evacuation time of $P[v_i, v_j]$ with the same sink for any j between i and n. See Fig. 2.2a. We assume the sink s is at v_j^+ and v_i evacuates to s along path $P[v_i, v_j]$. So, as per our above discussion about flow, we can say that all supplies from v_i to v_j move to s for evacuation. To determine the evacuation completion time we need to make sure that every vertex supply can reach the sink. If any supply is disrupted by the other vertices supply that also goes to the same direction to evacuate then the *congestion* occurs. Let v_h be that most distant vertex for which the congestion takes place, where $i \leq h \leq j$. The vertices from $V[v_i, v_{h-1}]$ have to wait on the intermediate vertex v_h until the supply of v_h evacuate completely. We call the vertices $V[v_i, v_h]$ generate a *cluster* and the vertex v_h is the *cluster head.*

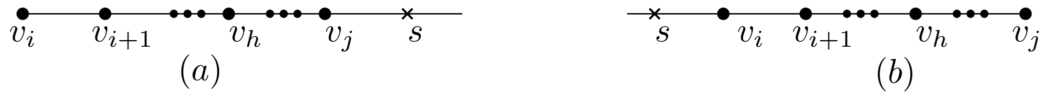

Figure 2.2: Evacuation when s is at: (a) v_j^+; and (b) v_i^-

Definition 2.2. (Cluster). If the supplies of a set of vertices have to wait at a transitional vertex before moving to a sink, then the set of vertices including the 'transitional vertex' is called the cluster.

Definition 2.3. (Cluster head). A cluster head is the vertex of a cluster C whose demand does not have to wait for evacuating to some sink. In other words, the supply of the cluster head of a cluster will be the first to reach a sink.

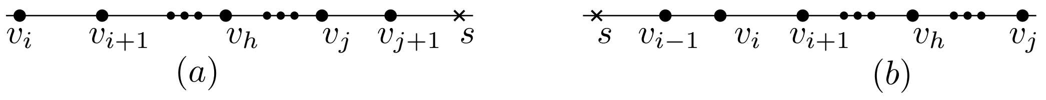

Figure 2.3: Evacuation when s is at: (a) v_{j+1}^{+}; and (b) v_{i-1}^{-}

Now we discuss another example where the sink moves. Let consider the Fig. 2.3a be the next step of Fig. 2.2a, meaning the current sink position is at v_{j+1}^{+} in Fig. 2.3a. We already determined the cluster and cluster head for the subpath $P[v_i, v_j]$ when the sink was at v_j^{+}. After moving the sink to the next edge, we need to determine whether the cluster information would be the same as it was before or should update. If the supply of vertex v_h needs to wait at v_{j+1} then the previous cluster head v_h should be updated with v_{j+1}; otherwise, the cluster and cluster head information would remain same. Similarly, we can move the sink at v_{i-1}^{-} as in Fig. 2.3b and determine the cluster information as above.

So now we can calculate the evacuation time for $V[v_i, v_h]$ by the following equation [7]:

$$\theta_L(s,[v_i,v_h]) = d(v_h,s)\tau + \frac{W[v_i,v_h]}{c(v_h,s)} \quad \text{for } s = v_j^{+} \tag{2.2}$$

Here, $d(v_h,s)$ (resp. $c(v_h,s)$) is the distance (minimum capacity) from the cluster head to the sink s, τ is the transit time per unit distance and $W[v_i,v_h]$ is the sum of weights from vertex v_i to the cluster head v_h. We denote by $\theta_L(s,[v_i,v_h])$ the *left time (or L-time)* for evacuation to a sink s and the supplies are coming from the left side of this sink.

Now consider Fig. 2.2b, where the sink is at v_i^-. By the similar way we can determine the cluster and cluster head and in this case, the evacuation time is

$$\theta_R(s,[v_h,v_j]) = d(s,v_h)\tau + \frac{W[v_h,v_j]}{c(s,v_h)} \text{ for } s = v_i^- \tag{2.3}$$

So, as per above discussion, we call $\theta_R(s,[v_h,v_j])$ is the *right time (or R-time).*

Lemma 2.4 ([1, 15]). *Given a subpath $P[v_i,v_j]$ of a dynamic path network P and a sink $s \in P[v_i,v_j], \Theta(s,[v_i,v_j])$ is represented by the following formula:*

$$\Theta(s,[v_i,v_j]) = max\left\{ \max_{v_h \in V[v_i,s]} \theta_L(s,[v_i,v_h]),\ \max_{v_h \in V[s,v_j]} \theta_R(s,[v_h,v_j]) \right\} \tag{2.4}$$

2.3 Basic concepts of evacuation

For a cycle C, if we are given a sink and a split edge, then the cycle can be represented as a path. We can compute the evacuation time of a cycle using the existing path algorithm as follows.

Fig. 2.4 illustrates two example path networks consisting of one and two edges, respectively. In Fig. 2.4a, let a sink s be placed on vertex v_1. As per our model definition, the weight of v_1 can evacuate instantly as it is the sink vertex. Therefore, we have only one vertex v_2 to consider for evacuation and needs to travel the distance $l = d(v_2,v_1)$. The first unit of supply takes τl time to travel from v_2 to sink s (as assumed on v_1). We assume the capacity of edge $e_1 = (v_1,v_2) = (v_2,v_1)$ is c that means maximum c amounts of supply can enter the edge per unit time. The total amounts of unit that needs to move is, $\frac{w_2}{c}$, where w_2 is the weight of vertex v_2. After first unit of supply entering into the edge, the second unit supply can enter the edge and so on. So, the total time requires for evacuating to the last unit of supply can be computed by $R-$time equation (2.3), that is, $\theta_R(s,[v_2,v_1]) = \tau l + \frac{w_2}{c}$.

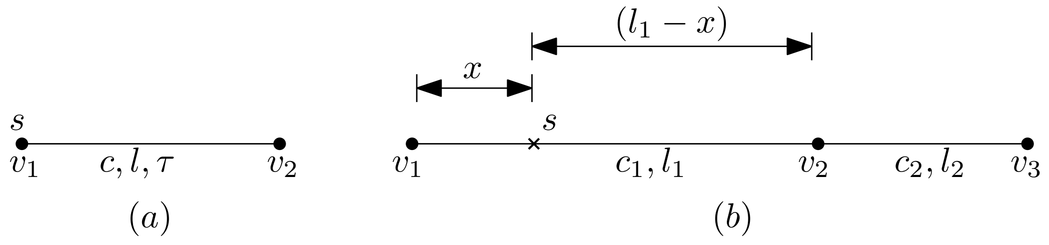

Figure 2.4: Example path networks with : (a) single edge, and (b) two edges.

Now, let us consider a bit complex path as in Fig. 2.4(b) consists of three vertices v_1, v_2, v_3 and edges $e_1 = (v_1, v_2)$, and $e_2 = (v_2, v_3)$. We also let a sink s be placed on edge e_1, which is x unit of distance apart to the right from v_1. So, we can write $d(v_1, s) + d(s, v_2) = l_1$. In this case, the supplies from both the left and right side of the sink s travel for evacuating at s.

For the supply at v_1 (resp. v_2), needs to move x (resp. $(l_1 - x)$) distance to the right (resp. left) for getting the sink s. As we discussed for single edge network, the evacuation time can be determined by the $L-$time (resp. R-time) equation (2.2) (resp. equation (2.3)), which is $\theta_L(s, [v_1, s]) = \tau x + \frac{w_1}{c_1}$ (resp. $\theta_R(s, [s, v_2]) = \tau(l_1 - x) + \frac{w_2}{c_1}$).

The supply of v_3 needs to cover l_2 and $(l_1 - x)$ distance to reach the sink. At first, the supply of v_3 has to come at vertex v_2 by spending $\tau l_2 + \frac{w_3}{c_2}$ unit of time and then from v_2 to the sink s it takes another $\tau(l_1 - x) + \frac{w_3}{c_1}$ time. Therefore, the total time requires, $\theta_R(s, [s, v_3]) = \tau(l_2 + l_1 - x) + \frac{w_3}{c(s, v_3)}$.

But if the supply from v_3 arrives at v_2 before w_2 has left completely towards the sink s, that means $\tau l_2 < \frac{w_2}{c_1}$, then congestion will be occurred at v_2. As a result, the supply coming from v_3 needs to wait at v_2. Then we say vertices v_2 and v_3 together form a *cluster* and in this case, vertex v_2 is the *cluster head* of the cluster.

In that situation, total weight at v_2 is $(w_2 + w_3)$ and the total evacuation time does not depend on τl_2 anymore. So, the total time to reach the sink for last unit of supply from v_3 is,

$$\theta_R(s, [v_3, s]) = \tau(l_1 - x) + \frac{w_2 + w_3}{c(s, v_3)}$$

The similar calculation can be applied for a more extended path network.

Let we are given a cycle C as shown in Fig. 2.5a and asked to find out the optimal solution for placing a sink. Our idea is to split all edges of the cycle network one by one. For the simplicity, let split an edge $e_i \in E$ for which the path $P[v_i, v_{i+1}]$ is generated as in Fig. 2.5b. For any split edge, the flow determines the path in the cycle and using the path algorithm we identify an optimal sink for that specific path. Then, the optimal split edge corresponds to the minimum value, so a brute force cycle algorithm calculates the solution by using the path algorithm $O(n)$ times for every split edge.

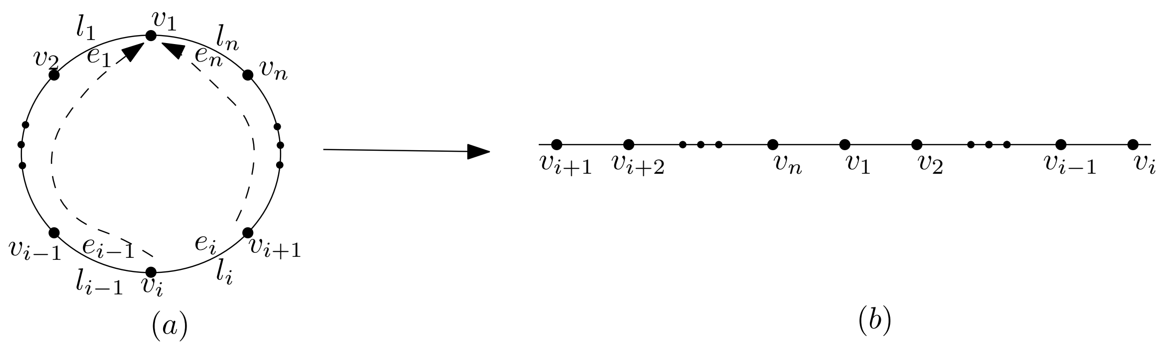

Figure 2.5: (a) A cycle network. (b) For splitting edge $e_i = (v_i, v_{i+1})$, the corresponding path network.

Now, we discuss with an example how to determine the optimal splitting edge. Fig. 2.6 illustrates an example cycle network. We are given a network consisting of six vertices $(v_1, v_2, ..., v_6)$ and their corresponding weights are $5, 2, 4, 3, 7, 9$ respectively. There are also six edges ($e_1 = (v_1, v_2), e_2 = (v_2, v_3)$, and so on) and the edge lengths are $6, 8, 12, 5, 2, 3$ accordingly.

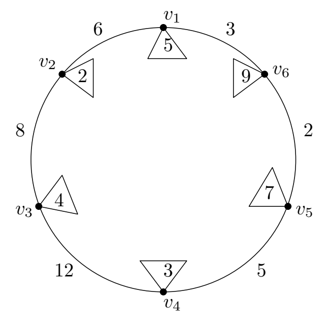

Figure 2.6: An example cycle network, C

Let us assume the sink is on v_1. For the simplicity of our discussion we consider the capacity for all edges are $c = 1$ and the transit time $\tau = 1$. Using equations (2.2) and (2.3) let compute the evacuation time of v_2 for both directions to reach the sink s placed at v_1

$$\theta_L(1,[2,1]) = 6 \times 1 + \frac{2}{1} = 8$$
$$\theta_R(1,[2,1]) = 30 \times 1 + \frac{2}{1} = 32$$

Where, $\theta_L(1,[2,1])$ and $\theta_R(1,[2,1])$ denote the evacuation time for clockwise and counter clockwise direction from v_2 to v_1 (the sink).

Similarly, for all vertices we compute the evacuation times and store them in two different arrays. Table 2.1 and Table 2.2 represent the evacation time for every vertex $[v_2, v_6]$ to the clockwise and counter clockwise directions, respectively.

Table 2.1: Evacuation time for all vertices in clockwise direction

Vertex:	2	3	4	5	6
Evacuation time:	8	18	29	38	42

Table 2.2: Evacuation time for all vertices in counter clockwise direction

Vertex:	2	3	4	5	6
Evacuation time:	32	26	13	12	12

So, now for every vertex we have two different evacuation times. We determine the larger one and store them in $\Theta(1, [v_2, v_6])$. For example, vertex v_3 has the evacuation time in counter clockwise direction 26 and in clockwise order 18. So, we store the evacuation time which we get for anti clockwise direction into the $\Theta(1, [v_2, v_6])$. In this technique we will find the following values for $\Theta(1, [v_2, v_6])$. See Table 2.3.

Table 2.3: Minimum evacuation time in either direction for all vertices

Vertex:	2	3	4	5	6
Evacuation time:	32	26	29	38	42

As we can see, the minimum value in $\Theta(1, [v_2, v_6])$ is 26 and comes for v_3 when the supply of v_3 moves to the sink in the counterclockwise direction. From the definition of confluent flow in sub-section 2.2.1, we can ensure that all vertices $V[v_3, v_6]$ will move towards the sink in a counter-clockwise direction and vertex v_2 only move to the clockwise direction. Edge e_2 will not be used, meaning, in this case, the split edge is, e_2. So, from our given cycle C by splitting edge e_2 we form the following path.

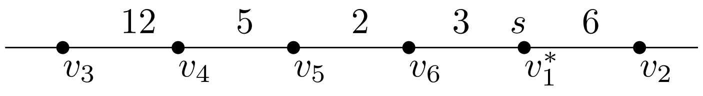

Figure 2.7: An example path obtained from given cycle C, by splitting edge, $e_2 = (v_2, v_3)$

So, for the sink location at v_1 and the split edge e_i we can determine the evacuation time $\Theta(v_1, (v_i, v_{i+1})$ using the equations (2.2) and (2.3) as described in [9]. For the single sink location problem on path $P[v_i, v_{i+1}]$ the algorithm requires $O(n)$ time, where n is the number of vertices on the path. Then we split the next edge e_{i+1} for the same sink location. As per our model definition, there are n edges in the cycle. So, we determine a total n number of evacuation times. The edge attaining the minimum value determines the optimal cost for placing single sink in the cycle network we call this edge the *optimal splitting edge* for that sink location. We move our sink in the next vertex v_2 and determine similarly another optimal splitting edge and so on. We observe that, if the sink location moves in counter-clockwise direction then the split edge either moves along the anti-clockwise direction or would be in the same position. Among all of these optimal splitting edges, the minimum would be our optimal evacuation time and the sink location for which we get this minimum is the optimal sink location.

As we mentioned earlier that the sink could be placed on edges too, so we need to compute the optimal evacuation time and sink location. In this case, we assume the sink is on edge e_1 then the evacuation time for the supplies who moves clockwise direction is $\Theta(s,[v_1,s])$ and for the anti-clockwise direction is $\Theta(s,[s,v_1])$ for any splitting edge. We can find the optimal splitting edge by the above procedure. To find out the exact position of sink we know that the evacuation time for clockwise and counter-clockwise is equal. Otherwise, on one side the evacuation time would be greater than the other side, which is not an optimal case. So we can write a formal relationship that applies to all cases.

$$\Theta(s,[v_i,s]) = \Theta(s,[s,v_{i+1}]) \tag{2.5}$$

Then we move our sink on the next edge e_2 and apply the same process to compute the evacuation time and sink location and so on.

Lemma 2.5. *The brute force algorithm for placing a sink on cycle takes $O(n^2)$ time.*

Proof. In the brute force algorithm we determine the evacuation time when the sink moves every vertex one by one. For every sink position, to find out the splitting edge, we spend $O(n)$ time. Then for every splitting edge, we compute the evacuation time to evacuate on the sink which requires another $O(n)$ time. So, the whole algorithm runs in $O(n^2)$ time. □

2.4 An optimal algorithm for 1 sink location problem

In the previous section, we analyze the brute force approach for placing a single sink on cycle which runs in polynomial time. We believe that the steps of the brute force algorithm can be implemented in amortized linear time by an appropriate data structure.

Our idea of the optimal algorithm is; first we first split any arbitrary edge e_i and construct a path. Then using the algorithm from [9], we compute the optimal solution for the $1-$sink problem on the path. Now, we move the leftmost vertex to the end of the path and treat it as a new path. We notice that the optimal sink for the new edge must be located to the right of the previous position. Again, the supplies in the right side of the sink do not change dramatically, just only a new vertex has been added. All we need is to know the cluster head information for the newly joined vertex. If we spend more than constant time to find out the cluster head information, then our algorithm cannot run in $O(n)$ time. To this end, we propose a data structure to store the cluster head information called *cluster head forest* (CHF for short) which we can use to obtain the evacuation time for the sub-paths in constant time for uniform capacities and amortized $O(\log n)$ time for general edge capacities. In the next chapter, we are going to introduce our CHF. The power of this data structure is to process batches of queries efficiently. The algorithm has two phases: preprocessing and sink placement. In the first phase we construct the CHF, and in the next stage, we place the sink in an optimal location on the path.

Chapter 3

The Cluster Head Forest

3.1 Introduction

In this chapter, we introduce our proposed data structure which plays a central role in our claimed results. In graph theory, a *forest* is a collection of trees (possibly undirected) and a *tree* is a group of nodes connected by directed (or undirected) edges. We represent the cluster head (see definition 2.3) information for a given path by the topology of the forest. That is why we named our data structure as the *cluster head forest* (CHF). The CHF is constructed for both uniform and non-uniform edge capacities in this chapter. We denote by C_i and S_i the clusters and sections of a cluster, respectively throughout this chapter. Recall that in chapter 2 we assume S is the set of k-sinks. These are two different notations they do not have any relation with each other.

For the simplicity, first, we consider creating a CHF for uniform capacity in section 3.3. Before that, properties of the CHF is discussed in section 3.2. For better understanding, we also construct a CHF for an example path. In section 3.4, we show how to extend the CHF for uniform capacity to the arbitrary capacity case. We also discuss two complications that need to be handled for general edge capacities. Section 3.5 comprises using of the CHF for answering both a single query and a set of queries.

The cluster head forest (CHF), represented by $\mathcal{F}$, which we define for a path graph. Let us given a path $P[u_1, u_q]$ as shown in Fig. 3.1 and we want to determine the evacuation time for any arbitrary pair (i, j) with $1 \leq i \leq j \leq q$, when the sink s is on edge $e_{i-1} = (u_{i-1}, u_i)$. To do this, we need to find out a specific cluster that contains the supply of vertex u_j. In other words, we need the cluster head information of the farthest cluster from sink s that has the supply of u_j. Once we obtained the cluster head u_h, we can compute the left and right evacuation time using equations (2.2) and (2.3), respectively.

3.2 Properties of the CHF

- Every vertex on the given path $P[u_1, u_q]$ is also present in the forest as a CHF node. We use the term *nodes* to refer to the nodes of the CHF, to distinguish them from the vertices that are on path P.

- We construct the CHF for a path $u_1, u_2, \ldots, u_q$ and the direction of the evacuation flow is from right to left, i.e., $u_q, u_{q-1}, \ldots, u_1$.

- We consider positions u_i^- for all u_i, and all of the supply between u_i and u_n evacuate to u_i^-. We observe that when the sink moves along an edge, the clusters cannot change. The reason behind this observation is that the clusters from the sequence for u_{i+1}^- still have to travel through the same edges and with a new edge. Because of the new supply of u_i or the reduced capacity of edge e_{i-1} two or more clusters can merge. So, for any given subpath $P[u_i, u_j]$ evacuating to u_i^- we can find the cluster containing u_j from the cluster sequence for the sink location of u_i^-. To determine the cluster head of that specific cluster, we can start walking from u_j towards u_i.

- In the CHF, at each node, we store the cluster head information. The subtree rooted at the node represents the cluster head.

- When we obtain the sequence for sink u_i^- recursively from the cluster of u_{i+1}^-, if two clusters merge, in the CHF we make the subtree of the cluster farthest from the sink a child of the cluster head of the cluster to the sink. All descendants of a subtree T_i, where $i \in \{1, \ldots, n\}$ belongs to the same cluster and the root ρ of that subtree T_i is the cluster head.

- In Our CHF construction process if we traverse the path from right to left (resp. right to left), then the index of the root is lower (resp. higher) than any other node index of a subtree. Similarly, the rightmost child of a subtree T_i contains the maximum index. (For arbitrary case, edges of the CHF trees need to be labeled with the sink location on the path)

3.3 CHF construction for uniform capacities

To create a cluster head forest, we traverse the path denoted $u_1, u_2, \ldots, u_q$, from right to left, i.e., from u_q to u_1 where q is the total number of vertices on the given path. See Fig. 3.1. As defined in [6], we can visualize this construction process by typical *cluster diagram* or *sequence diagram.* Now, consider the following cases:

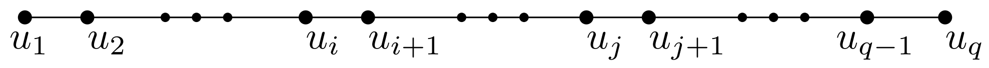

Figure 3.1: CHF construction on a path $P[u_1, u_q]$

- *Base Case:* Let the sink s be at u_q^- (recall that by u_i^-, we denoted a point on edge e_{i-1} that is arbitrarily close to u_i). We first check whether the leftmost cluster merges with the other clusters or not. If they do not merge, we are done; otherwise, we merge and check the conditions with the next cluster, and so on. In this sink position, we have only one vertex u_q in the right of s. Supply of vertex u_q evacuates to the sink s without any delay for the supplies of other vertices. For this reason, there is only one cluster C_1 and u_q is the cluster head. See Fig. 3.2a. The evacuation completion time is w_q/c, where w_q and c denote the supply of vertex u_q and the capacity of edge e_{q-1}, which is uniform for all edges, respectively. Therefore, the CHF consists of a single node q.

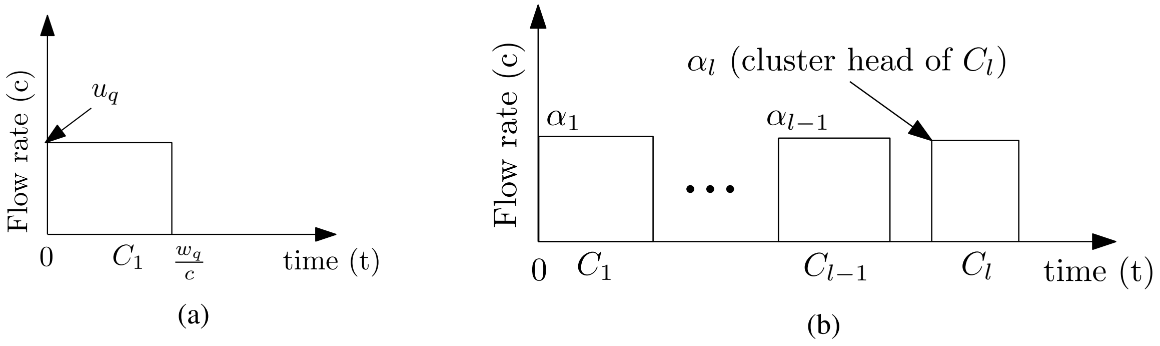

Figure 3.2: Cluster sequence diagram: (a) base case and (b) general case

- *General Case:* Suppose, the sink s is at u_{i+1}^-, $e_i = (u_i, u_{i+1})$ and the cluster sequence consists of clusters $C_l, \ldots, C_1$ when cluster C_l has cluster head α_l, shown in Fig. 3.2b. We now update the CHF by incorporating the node i. Let s is at u_i^-. We compute the evacuation time for all supplies from $u_i, \ldots, u_q$ to the sink.

  Here we have to consider two scenarios:

  (a) If $\tau l_p > w_{p+1}/c$, where $p \in \{i, q-1\}$, i.e., no congestion occurs at u_i. In other words, the cluster of u_i will not merge with the other clusters. Therefore, a new independent cluster will be added to the sequence. For the simplicity, let the newly formed cluster is C, where $\alpha = u_i$ is the cluster head.

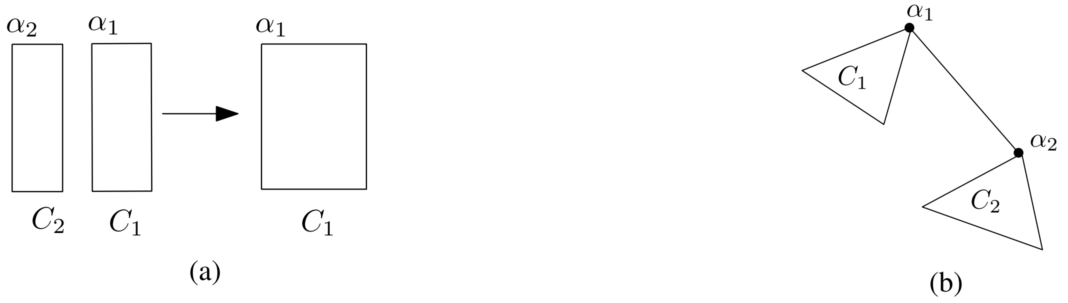

Figure 3.3: (a) Merging two clusters (b) two clusters (subtrees) adding by an edge in the CHF

  (b) If $\tau l_p \le w_{p+1}/c$, where $p \in \{i, q-1\}$, i.e., the supply already present at u_i has not left towards the sink s completely, but the supply from previous clusters arrive at u_i in the meantime. Then we can say that a new cluster is formed by joining some of the clusters from $C_1, \dots, C_l$ with the cluster head u_i. The total weight of the cluster needs to be determined, meaning we have to find out which clusters from subpath $[u_{i+1}, u_i]$ merge with the cluster, C. As we already know the congestion occurs at u_i, it implies that at least clusters C and C_1 merge. So we join them together. Once a cluster C merged with C_1, they will remain combined throughout the entire path traversing. For instance, see Fig. 3.3a. In the CHF, we represent these two clusters by adding an edge between them. For example, consider Fig. 3.3b which is a cluster tree in the forest and the parent node is α_1. Similarly, we examine the congestion between clusters C and C_r, where $h \in \{1, \dots, l\}$. The number of such checking at an iteration i can be at most $(q-i)$, where q is the total number of vertices on the given path. We continue this joining process until a cluster does not need to wait for evacuation. In Fig. 3.4a, C_{p-1} is such cluster. Once we got the cluster C_{p-1}, our merging process for the i^{th} iteration is done.

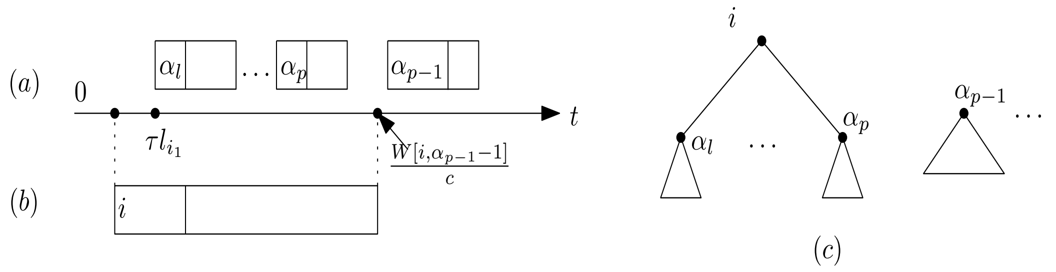

Figure 3.4: (a) The cluster sequence for the sink at u_1^- before incorporating the supply from u_i; (b) cluster sequence after clusters $C_{l+1}, C_l, \dots C_p$ have merged; (c) the corresponding CHF

Similarly, we assume a sink is at the left of every vertex $u_p \in (u_1, u_q)$ and perform merging process as discussed above. Finally, we generate a cluster head forest as drawn in Fig. 3.4c.

3.3.1 CHF construction for an example path

Now, we discuss with an example to generate the CHF from a path. Fig. 3.5 illustrates a path network which is consisting of three vertices (u_1, u_2, u_3) and their corresponding weights are $12, 3, 5$, respectively. There are two edges, $e_1 = (u_1, u_2)$ and $e_2 = (u_2, u_3)$ with lengths 2 and 4, respectively.

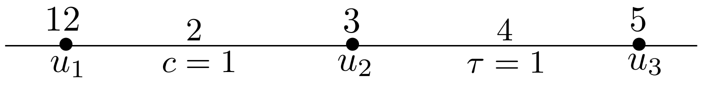

Figure 3.5: An example path network, $P[u_1, u_3]$

According to the base case of the CHF construction, let the sink s is at u_3^- (i.e., ε distance left from u_3). For the simplicity of our discussion, let $\varepsilon = 0.5$, both edges have the same capacity, $c = 1$, and the travel time per unit distance is $\tau = 1$. So, using equation (2.3) we determine the evacuation time for u_3, $\theta_L(u_3^-, [u_3, u_3^-]) = 5.5$. Therefore, in CHF there is only one node u_3 as shown in Fig. 3.7a and the sequence diagram is in Fig. 3.6a.

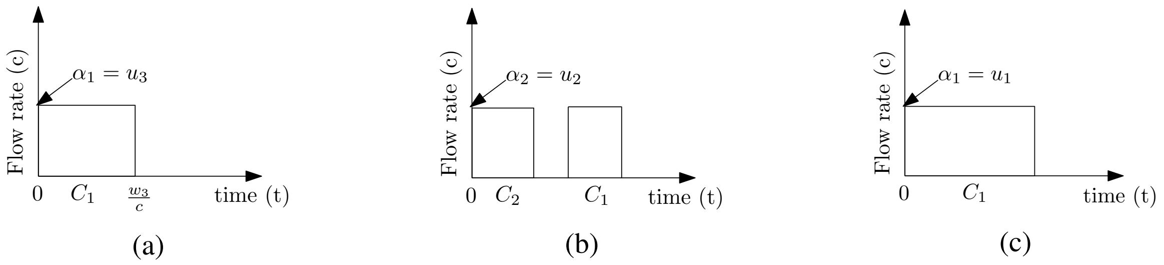

Figure 3.6: Sequence diagram for an example path: (a) base case, (b) second iteration, and (c) last iteration

In the next iteration, let the sink s at u_2^-. We compute the evacuation times for the supplies of u_2 and u_3 are $\theta_L(u_2^-, ([u_2, u_2^-]) = 3.5$ and $\theta_L(u_2^-, [u_3, u_2^-]) = 9.5$, respectively. As $\theta_L(u_2^-, [u_2, u_2^-]) < \theta_L([u_3, u_2^-])$, so there will be no congestion. We show the cluster sequence of this iteration in Fig. 3.6b. Hence, in CHF, there will be two disjoint nodes u_2 and u_3 as shown in Fig. 3.7b.

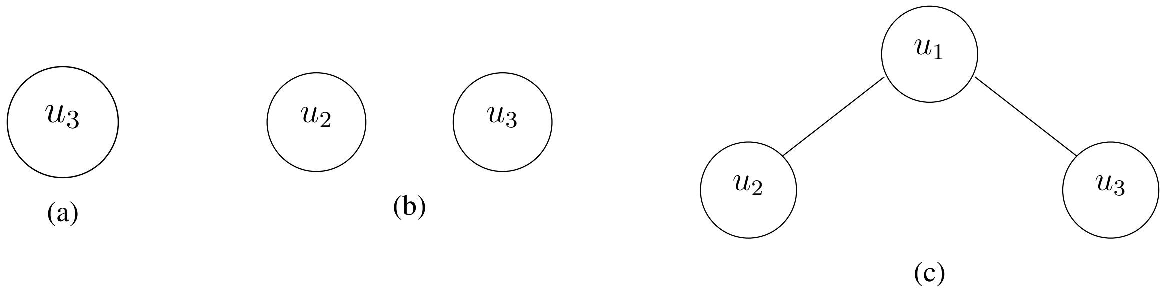

Figure 3.7: Cluster head forest construction steps for: (a) base case, (b) two disjoint clusters at 2^{nd} iteration, and (c) final CHF

Final iteration would be moving the sink s is at u_1^-. So, there is a new cluster C with cluster head $\alpha = u_1$. We calculate the evacuation times for (u_1, u_2, u_3) are $12.5, 5.5$, and 11.5, respectively. As we cam see the evacuation time for evacuees of u_1 is the largest among the times. So, congestion occur at u_1, which implies that at first cluster C and C_1 get merge into cluster C. In CHF, we connect the node u_1 and u_2 by an edge and represent them as a tree where the parent is u_1. Now we have two cluster C and C_2. Similarly, we compute the evacuation time for C and C_2, are 15.5 and 11.5. Again congestion occurs and C joins with C_2. So, there is only one cluster C with cluster head u_1. See Fig. 3.6c. In CHF, we add them up with an edge. We sketch the full cluster head forest in Fig. 3.7c.

Lemma 3.1. *A cluster head forest, $\mathcal{F}$, can be constructed from a dynamic flow path network in $O(q)$ time for uniform capacities, where q is the number of vertices on the given path.*

Proof. We construct the CHF by moving a sink at the left of every vertex. We traverse the whole path from u_q to u_1. So there are q iterations. At every iteration, in the CHF exactly one new node will be added. Then we check whether the newly added cluster can merge with another or not. For the positive answer of the checking, they will be added by an edge in the CHF. For one iteration, several checking may be needed. However, we check clusters at only once. Our observation is that once two clusters merge, they remain merged. Since we perform a check for every and every cluster that we merge, we have $O(n)$ verification steps. In the CHF there are at most q nodes. Therefore, we say that CHF can be constructed in linear time. □

3.4 Cluster Head Forest for arbitrary capacities

At first, we define some terms that are used throughout the CHF construction process for general edge capacities. In section 2.2, we defined the notions of cluster and cluster head. Now consider the following definitions

Definition 3.2. (Section). A *section* is the subinterval of constant rate. In other words, the largest slice of a cluster with same flow rate is called a *section*.

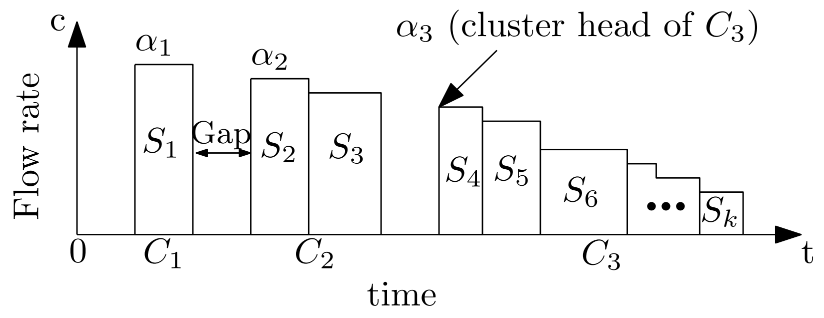

Figure 3.8: Sections in different kind of clusters. In the first two clusters C_1 and C_2, they have one and two sections respectively. Cluster C_3 consists of k sections.

Definition 3.3. (Simple cluster). A cluster that contains only one section is called the *simple cluster*. In Fig. 3.8, cluster C_1 is a simple cluster.

Definition 3.4. (Staircase cluster). If a cluster has two or more sections with different flow rate then we call it as a *staircase cluster*. Sections of this kind of cluster are arranged in descending order (i.e. the leftmost section has the largest flow rate). In Fig. 3.10, we represent the arrival rate of supply at the chosen sink. The arrival rate varies with time, therefore clusters C_2 and C_3 are *staircase clusters*.

Definition 3.5. (Gap). A *gap* or the empty space between two adjacent clusters is defined as a maximal interval of zero flow rate [5]. In Fig. 3.8, the gap is the space between cluster C_1 and C_2.

3.4.1 CHF construction

Our idea to construct the CHF for arbitrary capacities is the same as for uniform capacities. We start to compute cluster sequence with the case $i = q$, then $i = q-1$ and so on. We continue this computing for all sub-intervals $P[u_i, u_q]$ when the sink is at u_i^-. Because of the general edge capacities, the cluster sequence consists of sections with different flow rate.

Properties 3.6. In the arbitrary capacity case, the cluster sequence consists of sections with non-increasing flow rate if the sections are ordered according to increasing start time.

Proof. For the different edge capacities, the height of simple clusters would also be different. When two simple clusters with different flow rate get to merge, they combinedly form a staircase cluster. The left cluster has more height that is why it needs more time to evacuate, and in the meantime, the supplies from the right clusters have arrived. Similarly, when a simple cluster merges with a staircase cluster with two sections and then there should be a staircase cluster with three sections where the leftmost sections have more height, and it has the smallest start time. □

General case: Recalling the general case of CHF construction for uniform capacity. Suppose, the sink is at u_{i+1}^- and the clusters $C_l, \ldots, C_1$ with cluster head $\alpha_l, \ldots, \alpha_1$, respectively, are present in the cluster sequence, where $\alpha_l = i+1$.

We now move the sink at u_i^-. For these settings, let the supply of u_i generates a new cluster C_{l+1} to the cluster sequence. It is possible that sections with higher rate need to be flattened and therefore clusters a merge at the front because of the weight of vertex u_i, but also anywhere because sections can get flattened for a smaller capacity e_{i-1}. See Fig. **??**

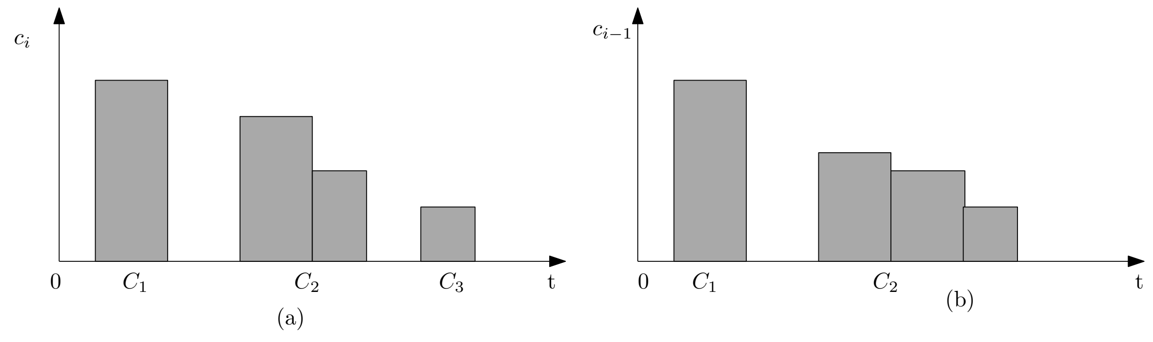

Figure 3.9: Sections flattened at middle of the sequence.

We need to find out as soon as possible which clusters merge because of the supplies of u_i, and which merge because of the capacity e_{i-1}. We can determine these events by the notion of *critical capacity*.

Definition 3.7. (Critical capacity). The critical capacity is the maximum capacity at which two adjacent sections (or a section and a gap) of different flow rate can merge.

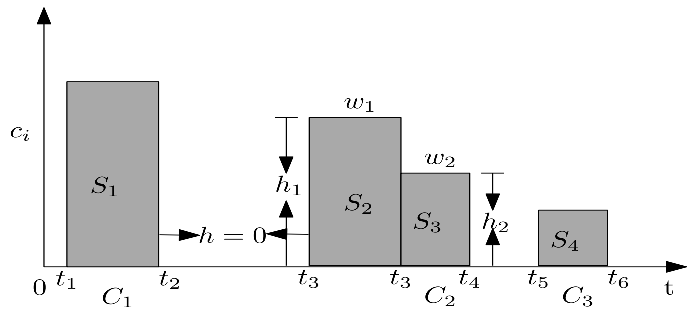

Figure 3.10: critical capacity

To compute the critical capacity of sections S_2 and S_3 of Fig. 3.10, let the height (flow rate) and weight pairs of S_2 and S_3 are (h_1, w_1) and (h_2, w_2), respectively. So, the critical capacity should be less than h_1 but higher than h_2. Therefore, the critical capacity would be

$$(h_1 - \phi_c)\frac{w_1}{h_1} = (h_2 + \phi_c)\frac{w_2}{h_2} \tag{3.1}$$

Where, $h_1 > h_2$ and ϕ_c denotes the critical capacity. Let $\frac{w_1}{h_1} = \Delta_1$ and $\frac{w_2}{h_2} = \Delta_2$ then the equation (3.1) can be rewritten as

$$\phi_c = \frac{w_1 - w_2}{\Delta_1 + \Delta_2} \tag{3.2}$$

If there is a gap between two sections (e.g. in Fig. 3.10, consider the section S_1 and S_2) the right term of (3.1) will be zero because a gap is equivalent to a section with zero rate, $h_2 = 0$. Similarly, we determine the critical capacity for every pair of adjacent sequences for the cluster sequence of $P[u_{i+1}, n]$, and Then we can process this cluster sequence to obtain for $P[i, n]$ as follows:

First, we examine the weight of u_i and compute the critical capacity ϕ_c between u_i and the first section from the sequence of $P[u_{i+1}, n]$. We insert the determined ϕ_c into a max-heap, $\mathcal{H}$, which consists of the critical capacities for every two consecutive sections or one section and a gap.

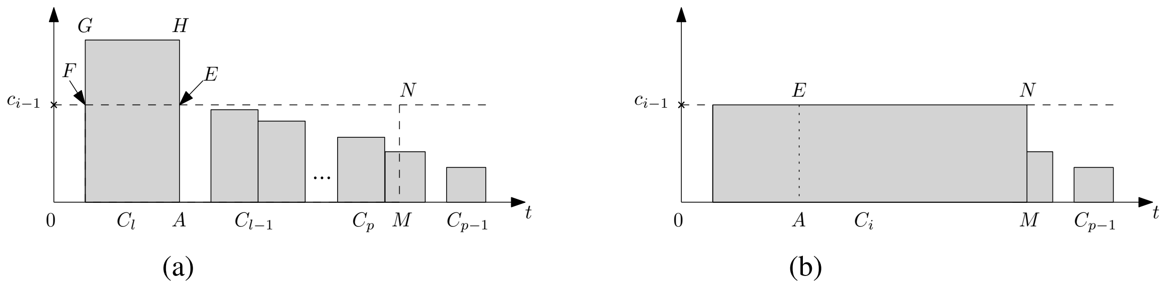

Figure 3.11: Because of the reduced capacity c_i, cluster C_l needs to be flattened. (a) the white areas between AE and MN can be filled by the gray parts of rectangle $EFGH$ of cluster C_l; (b) after merging the resultant cluster sequence

In the second step of the iteration, we pop up the root of the heap, $\mathcal{H}$. As we are using max-heap, that means the root has the maximum value of critical capacity in the heap, we denote this highest critical capacity value by $\phi_{c_{max}}$. So, now we have to consider two scenarios based on the value of c_{i-1} and $\phi_{c_{max}}$.

- Scenario 1: `No action is required.`

  If$(c_{i-1} \geq \phi_{c_{max}})$, i.e., if the capacity of edge e_{i-1} (since our sink is at u_i^-) is greater than the maximum critical capacity then all clusters $C_{l+1}, \dots, C_1$ can move to the sink without any disruption. Then we move our sink at u^-_{i-1} for a new iteration.

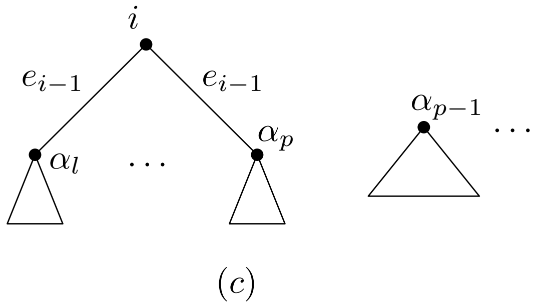

Figure 3.12: The CHF for aribitrary case at i^{th} iteration

- Scenario 2: `Flattening required`

Otherwise i.e., if$(c_{i-1} < \phi_{c_{max}})$. The maximum critical capacity value $\phi_{c_{max}}$ tells us to flatten the section(s) for which we get the corresponding $\phi_{c_{max}}$. That means, merging the corresponding sequences and updating the set of critical capacities because now we have a new merged section and the critical capacity of this new section must replace the critical capacity of the previous unmerged section. This maximum ϕ_c can be at any place of the sequence. Let the section of C_l and the gap, for example, have the $\phi_{c_{max}}$. In that case, cluster C_l needs to be flattened because the supplies of this cluster can not move to the sink with its actual capacity, at best they could flow with $\phi_{c_{max}}$ rate. In other words, the supplies of C_l will be spilled over to the rest of the sequence. See Fig. 3.11a. We let the supplies of rectangle $EFGH$ filled the all-white areas between the line AE and MN. More particularly, the sum of the white areas in the $AEMN$ rectangle equals the area of $EFGH$. We flattened the cluster C_l and the new height of C_l would be $AE = MN = c_{i-1}$. Because of this flattening let assume clusters $C_l, \dots, C_{p-1}$ and some portions of cluster C_l merged, as illustrated in Fig. 3.11b. How many clusters merge for one flattening can be determined as follows, if $\frac{W[u_i,u_r]}{c(u_{i-1},u_r)} \geq \tau d(u_i,u_r)$, then we two clusters merge, when $l \leq r \leq p$. We sequentially check whether a section needs to merge with cluster C_l or not. Once two clusters got merge it will remain merged throughout the traversing the path, that means we are not computing clusters merging more than $O(n)$. Here clusters merge, so we incorporate a new node in CHF i and make it as a parent of cluster nodes $\alpha_l, \ldots, \alpha_p$. We connect every child with parent i by an edge and label all edges as e_{i-1}. Look at Fig. 3.12. This labelling denote that i is the parent of these children as long as the sink is on the edge e_{i-1} or to the left (i.e., e_{i-2}, e_{i-3} and so on) of e_{i-1}. So, in general, we can say that, if any two clusters merge at any location of the sequence we add a new node in the CHF and make one of them as a parent of these clusters, would be the parent. The cluster head which has a lower index would be the parent node, and the other cluster head would be the children. For every child, we draw a connection to the parent and label the edge with the sink position for which two clusters merge.

In this case, we merged some full clusters $C_l, \ldots, C_{p-1}$ and the first section of C_p. But some supplies also spilled over S_2 of C_p. So technically this S_2 of C_p is now the second section of the newly formed big cluster C_i. The revised cluster sequence diagram is shown in Fig. 3.11b. So now for tracking the section(s) of a cluster, we store the following tuple of information at the cluster head of C_i, which is α_i

$$(s_i, h', T', \delta)$$

Here,
s_i denotes the sink location at i^{th} iteration.
T', represents the flow time where the next section begins. h' denote the flow rate, and δ is the portion of the supply of a vertex that flows with rate c_i. When cluster C_l merges with some clusters and the rate drops from c_{i-1} to the height of C_l. We can determine the area of section i with height c_i. Suppose, the area of C_i is W_i satisfies $W[i, q-1] < W_i < W[i, q]$. So, cluster C_l contains vertex q, and now cluster

C_l together with q is part of the new staircase cluster with head i. So, in this case, δ is the part of vertex u_q.

When two or more clusters merge, we update the information accordingly. To update the information we can easily get the value of the tuple. The sink location s_i is the position on edge for that particular iteration. The flow rate h' is the minimum capacity or height from the sink location to the particular section. We can find the δ using **??**. Let the S_2 of C_i is the supply of any vertex u_h. According to our description, we determine the tuple values to store in the α_i as follows

(a) s_i is the edge e_i.

(b) We can find the flow rate or height of the section by using a balanced tree introduced by [7]. Bhattacharya et al. introduced the critical capacities and upper envelope tree (CUE tree). By implementing a CUE tree, at every node of this tree, we store the left and right capacity functions which can tell us the minimum capacity between two points in $\log n$ time.

(c) Let $T' = x$, we solve x by the following equation

$$A_i = xh_1 + (T - x)h_2 \tag{3.3}$$

Where,

A_i is the total area (supplies from vertex u_i to u_h) of the cluster C_i.

T_i is the total transit time for C_i, which is $\tau d(u_h, u_i)$.

h_1 and h_2 is the height or flow rate of sections one and two respectively.

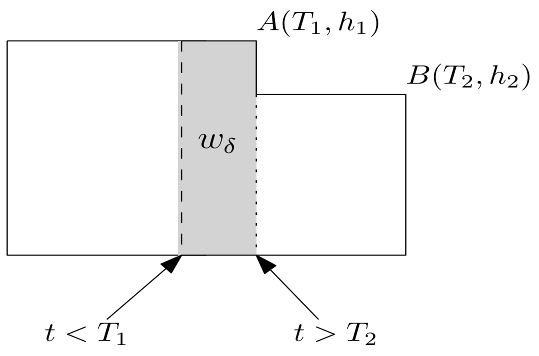

Figure 3.13: Determination the value of δ

(d) To find out δ, consider the Fig. 3.13

So, between $t < T_1$ and $t > T_2$, we perform binary search. We know the total area of the cluster and the vertex supply of u_h. Using (**??**), we can easily find out the w_δ

We get the next maximum value from the existing heap and perform above one of the two scenarios and so on. For a particular sink position, we continue this process until the first scenario executes.

Lemma 3.8. *For a given dynamic flow path network with general edge capacities, a cluster head forest $\mathcal{F}$ with a tuple of information can be constructed in $O(n \log n)$ time, where n is the number of vertices on the given path.*

Proof. To construct the CHF for the arbitrary capacity case, after determining the critical capacity (as discussed above) we insert them into a max-heap, and in later we pop the maximum value from the heap. This 'insertion' and 'pop' operations take $\log n$ time, separately. The other operations can be solved in amortized linear time as we did for uniform capacities. Therefore, we can claim that the construction of CHF for a path with general edge capacities takes $O(n \log n)$ time. □

3.5 Using the CHF

3.5.1 For a single query

For any given subpath $P[u_i, u_j]$ to determine the evacuation time $\Theta(u_i^-, [u_i, u_j])$ all we need is to know the information about the head of the cluster C that contains the supply of vertex u_j. Let the cluster C represent as a tree T_i with root $\rho = u_h$. We start traversing the CHF from node j towards the root ρ until we encounter an edge with an index less than i or we reached the root ρ. Let h be the node where we stop. Then h is the head of the cluster containing j. Now we can find the transit and waiting time constantly to compute evacuation time using (2.3), as follows

$$\theta_R(u_i^-, [u_i, u_h]) = d(u_h, u_i^-)\tau + \frac{W[u_j, u_h]}{c}$$

In arbitrary case, we also traverse the CHF in the same as we described above for the uniform case. However, in this case, we stop further traveling when we find out that the edge is not labeling with the sink position of lower index than e_{i-1}. Then using the following equation, we can compute the evacuation time.

$$\theta_R(u_i^-, [u_i, u_h]) = d(u_h, u_i^-)\tau + \frac{W[u_j, u_h]}{c(u_h, u_{i+1})}$$

In another case, if j belongs to a staircase cluster, then we need to find out the weight very carefully because the height and the weight of the sections are different. Recall that in the CHF construction process we stored a tuple of information in the head of every staircase cluster. Based on that information we can determine the weight. We can find the weight of any section by multiplying the flow time with the height.

3.5.2 Using of CHF for a set of queries

There are two types of set of queries. In one type we let the position of the starting a query i same, but we increase the ending position j of the queries or the vice versa. In the other type, we decrease either the starting or ending position of the queries. We formalize these queries as follows

- `Head_fixed_tail_increase` ($i = i'$ and $j = j' + 1$)
- `Head_increase_tail_fixed` ($i = i' + 1$ and $j = j$.)
- `Head_fixed_tail_decrease` ($i = i'$ and $j = j' - 1$)
- `Head_decrease_tail_fixed` ($i = i' - 1$ and $j = j'$.)

Now we explain this process with numerical values in the context of the above discussion. We enumerate every pair of vertices to generate a batch of queries, $1 \leq i \leq j \leq q$. To solve a batch of queries, we start with a query $(2,2)$ when the sink s is at u_2^-. We solve the query using the process we discussed for a single query. As we are willing to compute a set of queries, it is worthwhile to maintain two pointers i^* and j^* that indicates positions of vertex u_i and u_j in the CHF, respectively. Similarly, we solve the queries, $(2,3),\ldots,(2,q),(3,3),\ldots,(3,q)$ and so on. For a specific position of i, we are increasing the j values sequentially. If j^* reaches to the rightmost child of a tree, then in the following move j^* jumps into the next tree to the right. Thus, we do not visit the same node or edge twice for a specific position of i.

We also move the position of i one by one, and the queries are $(3,3),\ldots,(3,q)$. For every change of i^* in increasing order, we reinitialize the location of j with i^*, i.e., $i \in \{2,\ldots,q\}$ and $j \in \{i,\ldots,q\}$. Similarly, we can use the CHF for the second type of set of queries. However, in this case, the only complication is that the cluster can break down because of moving the sink from one place to another. Recall that when we add a tree in the CHF, we meant the sink is left of the root of that tree, which was the cluster on the path. So while we are moving the i to the right, then the cluster can break. However, in $O(1)$ time we can find out whether any node presents in a subtree or not.

Lemma 3.9. *If the CHF is available, we can answer a batch of queries in $O(q)$ time, where q is the vertices of the path.*

Proof. In above, we represented four separate types of queries. In these four cases, either we increase or decrease the position of j of any query (i, j), or we change the sink location. For every query, we need to use the process to answer the single query. To determine the result of a single query requires $O(q)$ time. When we increase or decrease the position of j, we can answer the query in $O(1)$ time based on the previous calculation. Similarly, we can also compute the query in $O(1)$ time when the sink position increases or decreases. □

3.6 Conclusion

In this chapter, we introduced our proposed data structure. We discussed the CHF construction process for both uniform and arbitrary capacities. Two challenges that one can face to use the CHF for answering queries also discussed. We also defined the properties of the CHF. For the redundant description, we omit to discuss the using of CHF for the arbitrary case because it would be the same as the uniform case with an extra $\log n$ factor. We showed how to answer a single query as well as a set of queries efficiently by using our CHF.

Chapter 4

Sink Location on Cycle Networks

4.1 Introduction

In this chapter, we present all of our algorithms for solving the sink location problem in dynamic cycle networks. First, we discuss the 1-sink problem. The convenient way to solve the multiple sink location problem is performing the feasibility test. For a given value, the feasibility test decides whether a problem instance is feasible or not and we discuss it elaborately in sub-section 4.3.1.

Throughout this chapter we consider a path $P[v_1, v_{2n-1}]$. We split an edge of the given cycle and construct a path with length $(2n-1)$ by combining two sets of vertices of the given cycle. We label the vertices of generating path as $v_1, v_2, \ldots, v_n, v_{n+1}, v_{n+2}, \ldots, v_{2n-1}$, where $v_{n+i} = v_i$ Fig. 4.1 illustrates the resultant path.

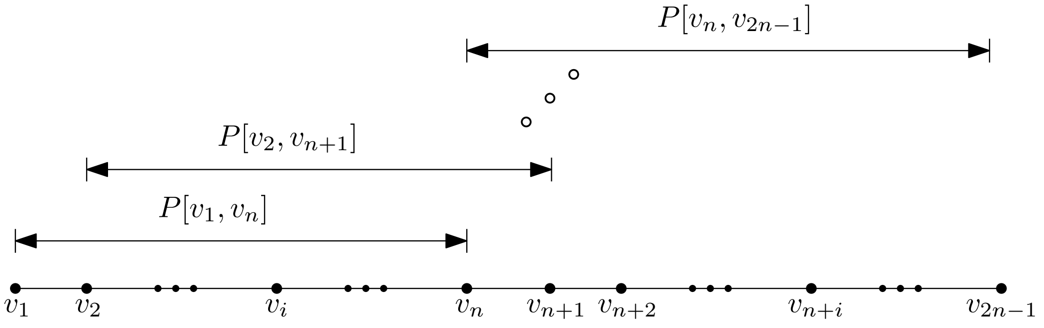

Figure 4.1: A path $P[v_1, v_{2n-1}]$ with length $2n-1$, where n is the number of vertices of the given cycle and $v_{n+i} = v_i$

4.2 Single sink problems

For the simplicity of discussion, let we split the edge e_n of given cycle C and produce the path $P[v_1, v_n]$. We start our algorithm placing sink s on e_{n-1} (or v_n^-). Now we have supplies at the left and right side of the sink s. To find out the evacuation time for this setting we need to know the cluster head information that contains the supply of v_1 and v_n for computing the L-time and $R-$time, respectively. In this regard, we use CHF for answering this queries as we described in subsection 3.5.1. Once we got the cluster head information by using equations (2.2) and (2.3) we can decide in which side of the sink has more supplies. We need to move our sink towards the larger value of these two evacuation times. To do that, we move our sink on the next edge to the left (in this case e_{n-2}). Using `Head_fixed_tail_increase` and `Head_increase_tail_fixed` queries (as described in subsection 3.5.2) we can find out the cluster head information for moving the sink. We continue this process until finding the exact position where the $L-$time and $R-$time are equal. Let e_i be our optimal sink location and the time needs to calculate evacuation time for this position is our optimal evacuation time for splitting the edge e_n as illustrated in Fig. 4.2.

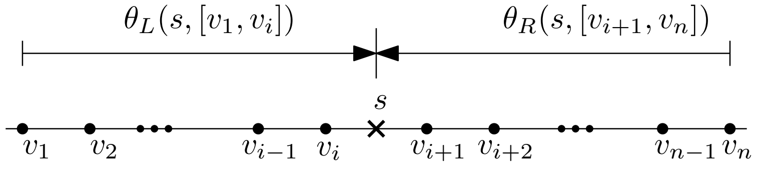

Figure 4.2: A path obtained from Cycle C by splitting edge e_n

As we discussed earlier, we need to split every edge and compute the evacuation time for each splitting. For this reason, we rotate the leftmost vertex v_1 to the end that generate a new path $P[v_2, V_1]$ (see Fig. 4.1) and also move the sink onto the edge e_{i+1}. By two simple checks, we can determine whether the cluster head changes for a new path or not. Now we make queries for $(i = v_2, j = v_{i+1})$ for left side of s and $(i = v_{i+2}, j = v_n)$ for right side of the sink s to the CHF for getting the cluster head. Again, we compute the optimal evacuation time for this splitting. Similarly, we determine the n number of evacuation times and a minimum of them is our final optimal evacuation time for the given path. We formalize our proposed algorithm for placing $1-$sink on cycle networks in algorithm 1.

Algorithm 1: Minmax $1-$sink on cycle algorithm

```
1  Input: C = (V, E), v[i] = vertices, e[i] = edges , $i \in \{1,2,...,n\}$;
2  Output: An optimal sink location; Optimal evacuation time of the solution;
3  Split any arbitrary edge $e_n = (v_n, v_1)$ of the given cycle $C$ and generate a path,
     $P(v_1, v_n)$. Here, for the simplicity of our discussion, we choose $e_n$ to cut, we split
     every edge to find out the optimal answer for the given cycle.
   // Sink placement phase
4  for $i \in \{1,...,n\}$ do
5      Let the sink $s$ be at $v_i^+$ Make a query for the left side of $s$ to the CHF and
         compute the L-time.
6      Similarly, calculate the R-time.
7      if $(L-time > R-time)$ then
8          move the sink one edge to the left
9          Repeat the computation of line 5 to 7 for new sink location
10     else
11         Find the exact location of sink s by solving equations (2.2) and (2.3)
12     Rotate $v_i$ to the end of rightmost vertex and construct a new path $P[v_i, v_{i-1}]$
```

Complexity analysis. The for loop in line 4 executes $O(n)$ iterations. We initialize a query inline 5 and 6, that gives us the cluster head information in $O(1)$. So, our algorithm requires $O(n)$ to determine the optimal sink location on the cycle.

Lemma 4.1. *Suppose the cluster head forest, $\mathcal{F}$, is available for dynamic cycle network with uniform edge capacities, we can solve the* $1-$*sink problem in* $O(n)$ *time.*

Proof. In our sink location problem, in the pre-processing phase, we construct the CHF by spending $O(n)$ as in lemma 3.1. We showed in the proof of correctness of algorithm 1 that it runs in linear time. Therefore, our 1-sink location problem algorithm finds out the optimal sink location in $O(2n) \approx O(n)$ □

Similarly, for an arbitrary case using Lemma 3.8, we can prove the following lemma as we spend $O(n \log n)$ time in both preprocessing and answering queries to the CHF for cluster head information.

Lemma 4.2. *Given a dynamic flow cycle network with arbitrary edge capacities, if CHF, $\mathcal{F}$, is available, the optimal 1-sink location and evacuation time can be computed in $O(n \log n)$ time.*

4.3 The multiple sink Problems

We perform $t-$feasibility test for solving the multiple sink location problem for both uniform and general edge capacity case.

4.3.1 The feasibility Test

Definition 4.3. (The $t-$feasibility test). We consider a problem instance is $t-$feasible if and only if every vertex supply can evacuate to a sink within time t and the total number of the sink should be less or equal k.

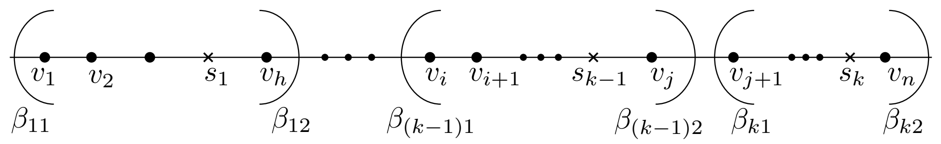

Figure 4.3: Feasible solution: all vertices are covered by a sink.

Figure 4.3 illustrates such a path $P[v_1, v_n]$ consists of n vertices, where left and right circular arcs represent the boundaries for sinks. A closed boundary denotes a subpath $P[v_i, v_j]$, which includes one of the each nearest left (β_{i1}) and right (β_{i2}) arc with respect to sink s_i. Here, the left (resp. right) border of sink s_i is denoted as β_{i1} (resp. β_{i2}). By boundary/border we mean, all supplies within the range $[\beta_{i1}\ to\ \beta_{i2}]$ will evacuate in sink s_i within time t. In every subpath $P[v_i, v_j]$ only one sink should be available. In total, we have $2k-$boundaries for $k-$sink: one left and one right boundary for every sink. If every vertex of the path belongs to any closed boundary then the multiple sink location problem is $t-$feasible otherwise infeasible.

We start to perform the feasibility test assuming the leftmost boundary is at vertex v_1 and then find the edge e_i containing the first sink s_1 so that the evacuation time for all supplies from v_1 to s_1 equals to time t. Then from s_1 we determine the right boundary β_{12}. We categorize our feasibility test in the following two stages. Step by step explanation for boundary selection and $t-$feasibility test are as follows:

- Step 1: (Original step)

  We start the original step by assuming the left boundary β_{11} of first sink s_1 is just left of vertex v_1. Now we determine the sink location as far as possible from v_1. To do that, we place our sink s_1 on just right after v_1 (i.e., v_1^+) and compute the $L-$time for the supply of vertex v_1. If the supply can move to s_1 within time t then we move the sink to the just right of next vertex (v_2^+) and calculate the $L-$time again and so on. After few repetitions, let consider our sink is at (v_h^+) and either some supplies can not move to the sink within time t or all left supplies can evacuate within exact time t. In first case, we place our sink on edge $(e_h = [v_{h-1}, v_h])$. We try to place sink as right as possible by solving the equation (2.2), where $\theta_L = t$ then we find the distance $d(v_1, s)$. In later case, the sink will be on v_h^+, which is ε distance to the right of v_h. Now as the sink is available on the path we have to find the right boundary β_{12}. Unlike the above process for placing sink now we need to consider all vertex supplies from the right side for evacuation to the sink s_1. Let consider s_1 is on v_h or edge e_h, we compute the evacuation time for all vertices $v_p \in [v_{h+1}, v_n]$. If the supply of v_p can not move to sink within time t then the β_{12} will be v_{p-1}.

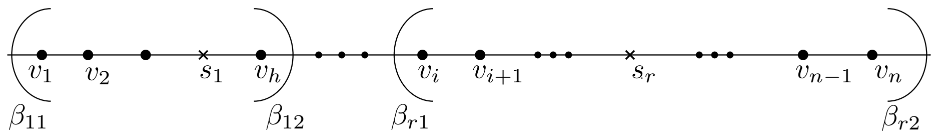

Figure 4.4: Feasible solution: all vertices are covered by a sink, where $r < k$.

Similarly we place $k-$sinks and determine their boundaries. Algorithm 2 explains the procedure for placing a sink when starting vertex v_1 and time t is given. If all supplies in the networks can evacuate in any sink within time t then the problem is feasible. In fact in two ways the feasibility test can be positive. If we need to place exact $k-$sinks or strictly less than $k-$sinks to cover all vertex supplies then in both cases the path is feasible. Consider Fig. 4.3 where exactly $k-$sinks are placed and the Fig. 4.4 where all supplies can reach one of the $r-$sinks, $r < k$. Until now we actually describe the feasibility test for a path [7] that requires $O(n)$ time. If it is not feasible then immediately we can not claim that the feasibility test is negative because it might be our splitting edge of the cycle C was not correct. Therefore, we need to split another edge and so on as described in step 2.

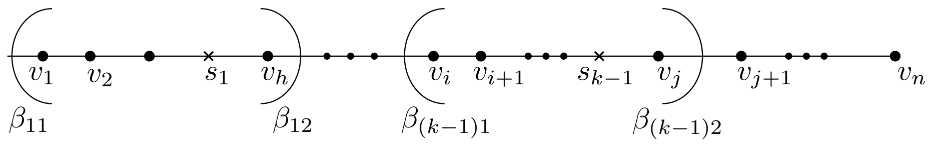

Figure 4.5: Last sink is the decision maker about the test

- Step 2 :

  Now we move the β_{11} to the left of the next vertex (e.g., $v_2, v_3, ...$) of previous iteration(s). We do not want to overwrite the boundary names of the original step. So, let β^k_{i1} and β^k_{i2} are the left and right boundaries for sink s_i, where k denotes the iteration number. In every iteration, we find the first sink location and then the right boundary for sink s_1 and so on by the similar manner as we described in Step 1.

Algorithm 2: $t-$feasibility(v_1, k, t)

```
Input: $v_1$ = Starting vertex, $t$ = time.
Output: 'Yes' for feasible solution, otherwise 'No'
/* We store sink locations in $s[i]$ and boundaries in $b[i]$. */
for $i \in \{1, \ldots, n\}$ do
    Place the sink $s$ on just right after $v_i$. So, $x[i] = v_i^+$
    Compute $L$-time for all left evacuees
    if (L-time$>t$) then
        Optimal sink location, $s[i] = v_{i-1} + \varepsilon$; Break;
    else if ( L-time$=t$) then
        Optimal sink location, $s[i] = v_i + \varepsilon$; Break;
/* We assume the left boundary $\beta_{i1}$ for the sink $s_i$ is on just
   left of $v_1$. Now We determine the right boundary $\beta_{i2}$. */
for $i' \in \{x[i], \ldots, n\}$ do
    Place the boundary $\beta_{i'2}$ on just right after $v_i'$. So, $\beta_i'2 = (v_i' + \varepsilon)$
    Compute $R$-time for all right evacuees
    if (R-time$>t$) then
        Boundary location, $b[i'] = v_{i'-1} + \varepsilon$; Break;
    else if ( R-time$=t$) then
        Boundary location, $b[i'] = v_{i'} + \varepsilon$. Break;
if (length of $x[i] \leq k$) then
    return 'Yes'
else
    return 'No'
```

For any sink s_i, if we have β_{ij}^l (l^{th} iteration) = β_{ij} (original step), then, the boundaries will be the same as in the original step, except that now we have an extra sink to cover the portion from β_{k2}. That means, we can reach the rightmost boundary β_{k2} of original step by placing $(k-1)$ sinks in the current iteration. We can check whether the supplies of the uncovered area can reach the extra sink within time t or not. If not, we do not need to continue moving the split edge because the uncovered portion only gets larger.

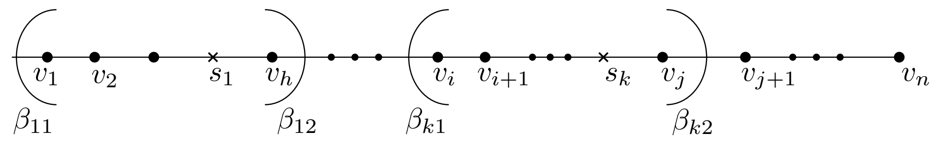

Figure 4.6: An infeasible solution because $V[v_{j+1}, v_n]$ are uncovered.

Let, the uncovered vertices were $v_{j+1}, \ldots, v_n$ as illustrated in Fig. 4.5. In other words, the supply of these vertices could not evacuate to any sink. In that case, we test whether the subpath $P[v_{j+1}, v_n]$ can be solved by placing k^{th} sink of this iteration or not. If the test result is 'yes' then the whole problem is feasible, otherwise not. See Fig. 4.6 for an infeasible solution.

Lemma 4.4. *If the cluster head forest, $\mathcal{F}$, is available we can perform the feasibility test for uniform capacities in $O(n)$ time, where n is the vertices of the given path.*

Proof. We already proved that the CHF construction requires linear time for uniform capacities as in Lemma 3.1. We showed in step 1 of our feasibility test algorithm that it can run in $O(n)$ time. Similarly, we argue that step 2 can also be done in linear time. So, overall our proposed $t-$feasibility test needs $O(n)$ time. □

Similarly, by Lemma 3.8 and in the consequence of Lemma 4.4 we can also prove the following Lemma.

Lemma 4.5. *For any given dynamic path with n-vertices, suppose the cluster head forest, $\mathcal{F}$, is available. We can test its feasibility for arbitrary capacities in $O(n \log n)$ time.*

4.4 Conclusion

In this chapter, we presented the single sink algorithms for uniform and arbitrary capacities. We also discussed the feasibility test algorithm in details with various cases and figures. Then based on the feasibility test we claimed two lemmas for above mentioned both types of edge capacities.

Chapter 5

Optimization

5.1 Introduction

The goal of this chapter is to optimize the algorithms that we obtained in the previous chapter. Here, we discuss two different optimization frameworks and present several algorithms for both uniform and general edge capacities.

5.2 Sorted matrix approach

In section 3.5, we discussed how to compute a batch of queries using the CHF. Recall that we enumerated every pair of integers i and j such that $1 \leq i \leq j \leq 2n$ to determine the evacuation time. We store the result of these queries in a $2n \times 2n$ sorted matrix $M_{2n\times 2n}$ by the following way as in [7]

$$M[i,j] = \begin{cases} OPT(n-i+1,j) & \text{if } (n-i+1) \leq j \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

Where, $OPT(i,j)$ denotes the optimal evacuation time for placing single sink on a subpath $P[v_i,v_j]$. In a sorted matrix [11], the elements in each row and column are sorted in either increasing or decreasing order. So, for any arbitrary pair (i,j) entry in $M[i,j]$ exists for which the evacuation time is optimal for the whole path. In this regard, Benkoczi et al. [4] discussed to solve the k-sink location problem on cycle using the sorted matrices framework introduced by Frederickson and Johnson [12, 13] that implies the following

Lemma

Lemma 5.1. *[12, 13] Suppose $h(n)$ is the complexity of pre-processing phase, $g(n)$ is the computational time to obtain a specific element in the matrix, and the feasibility test can be done in $f(n)$ time. Then we can solve optimal $k-$facility location in $O(h(n)+ng(n)+f(n)\log n)$ time.*

Table 5.1: Summary of complexities in different tasks

Tasks	Uniform	Arbitrary
Feasibility test, $f(n)$	$O(n)$	$O(n\log n)$
Selecting $M[i,j]$, $g(n)$	$O(\log n)$ as in [7]	$O(\log^3 n)$ as in [7]
Pre-processing, $h(n)$	$O(n)$	$O(n\log n)$

Table 5.1 represents the complexities of CHF construction, feasibility test and finding a particular value in the sorted matrix for optimizing our algorithms. Note that, we determine the value of $g(n)$ from the analysis of [7], where the authors showed that they can find a particular element in a sorted matrix in $\log n$ time. Then we have the following theorem by Lemma 5.1

Theorem 5.2. *The $k-$sink problem in dynamic flow cycle networks with uniform edge capacities can be solved in $O(n\log n)$ time*

In the general edge capacity case, taking the values from the table 5.1 then the Lemma 5.1 thus implies

Theorem 5.3. *Given a dynamic flow cycle networks with general edge edge capacities we can be solve the $k-$sink problem in $O(n\log^3 n)$ time*

5.3 Parametric search approach

Lemma 5.4. *[2] If the feasibility can be tested in $\alpha(t)$ time, then the $k-$sink can be found in $O(h(n)+k\alpha(n)\log n)$ time, $h(n)$ is the preprocessing time.*

By Lemma 3.8 we showed that our CHF construction requires $h(n) = O(n\log n)$ time, and the feasibility test takes $\alpha(n) = O(n\log n)$ time. Therefore, Lemma 5.4 gives us

Theorem 5.5. *Given a dynamic flow cycle network with n vertices, we can obtain the optimal $k-$sink in $O(n\log n + kn\log^2 n)$ time.*

When the edge capacities are uniform, based on Lemma 3.1 the preprocessing time is $h(n) = O(n)$ and then applying Megiddo's main theorem in [20] to Lemma 4.4, we get

Theorem 5.6. *Given a dynamic flow cycle network with uniform edge capacities, we can find an optimal $k-$ sink in $O(n + kn\log n)$ time, where n is the number of vertices on the cycle.*

Based on the number of sinks we can quickly decide which framework will provide the better result. If the number of the sink tends to the given number of vertices, then it is better to use the sorted matrix approach; otherwise the parametric search approach.

5.4 Conclusion

In this chapter, we discussed two best-known frameworks: sorted matrix and parametric search frameworks. These two frameworks give us four different running times for multiple sink location problems on the cycle with uniform and general edge capacities.

Chapter 6

Conclusion

6.1 Introduction

In this chapter, we first summarize our thesis. Section 6.2 consists of the gist of different chapters. Then in section 6.3, we sketch the future research direction.

6.2 Summary of the thesis

In this thesis, we studied the sink location problem on cycle networks. We considered dynamic flow networks and proposed algorithms for both uniform and general edge capacities.

In chapter 2, we defined our network model and discussed preliminaries and terms that we used throughout the thesis. We depicted a small two edge network for providing the elementary concepts. We also presented our sink location algorithm template to determine the optimal sink location on the edges and vertices in this chapter.

Chapter 3 is about our proposed data structure. We first generated the Cluster Head Forest(CHF) for uniform edge capacity then extended for solving the problem with general capacities. We depicted how to use our CHF in order to determine the evacuation time. Our CHF can provide an answer for a query with an arbitrary starting and ending point as well as batches of queries. We also showed that the Cluster Head Forest has the potential to solve a set of queries more efficiently

In chapter 4, sink placement on cycle is discussed. We presented a single and multiple sink location algorithm for uniform edge capacity. When the edge capacities are general, we also provided two algorithms of the similar kind as uniform capacity.

We optimized our proposed algorithms in chapter 5. We used parametric search and sorted matrix approaches as the optimization framework. The later is the most standard framework for optimization.

6.3 Future Scope

We believe our contribution in this thesis is significant for further research in other network topologies and as well as in different types of location problems.

Our result will lead to solve the same sink location problem in more complex topologies such as unicycle, cactus graph to obtain better results. In a connected graph if there are any two simple cycles have at most one vertex in common, then the graph is called *cactus graph*. Again, a *unicycle* graph is a special kind of cactus when there is only one cycle in the graph. Unlike our defined cycle network in unicycle graph, some branches can be hanging from the vertices of the cycle. The hanging branches can be treated as tree networks. Using our results with existing most efficient tree networks [8], sink placement problem in unicycle networks can be solved. Once the results for unicycle graph is available then the problem for a cactus network can also be solved. Thus, our research can extend to solve the sink location problem in different topologies. Table 6.1 represents the current state of sink location problem in the general graph.

Table 6.1: Current state of optimal sink location problem in general graph

Graph name	Uniform capacity		Arbitrary capacities	
	$1-$sink	$k-$sink	$1-$sink	$k-$sink
Cycle[1]	$O(n)$	• $O(n\log n)$, and • $O(n+kn\log n)$	$O(n\log n)$	• $O(n\log^3 n)$, and • $O(n\log n+kn\log^2 n)$
Unicycle	open	open	open	open
Cactus	open	open	open	open

[1]We proposed in this thesis the above cycle results and all of them are described in chapter 4 and 5.

For solving the other kinds of sink location problems, the min-sum objective, for example, our proposed data structure can be used to design a powerful algorithm for determining the evacuation cost more efficiently.

We also believe it is possible to obtain a particular element in spending $O(\log^2 n)$ time from a series of sorted matrices. Then the time complexity of our proposed algorithm for arbitrary capacities can be improved by a $\log n$ factor.

6.4 Publication

We submitted the parts of our research [6] that we conducted for this thesis on the same topic in 11th *International Conference on Algorithms and Complexity (CIAC-2019)*. Click here to read the full paper (accessible only from the e-mail address of University of Lethbridge). Due to space constraints, we could not discuss the arbitrary capacity case in the paper. We have a plan to submit the paper to a journal with the detailed information.

6.5 Conclusion

In this last chapter of the thesis, we summarize our discussion of preceding chapters. We also presented the current state for solving the sink location problem in the general graph and with clear remarks, we showed the areas that are still open for research. We argued that our proposed data structure can be used to obtain results for other topologies.

Bibliography

[1] Guru Prakash Arumugam, John Augustine, Mordecai J Golin, Yuya Higashikawa, Naoki Katoh, and Prashanth Srikanthan. Optimal evacuation flows on dynamic paths with general edge capacities. *arXiv preprint arXiv:1606.07208*, 2016.

[2] Guru Prakash Arumugam, John Augustine, Mordecai J Golin, and Prashanth Srikanthan. A polynomial time algorithm for minimax-regret evacuation on a dynamic path. *arXiv preprint arXiv:1404.5448*, 2014.

[3] Nadine Baumann and Martin Skutella. Solving evacuation problems efficiently–earliest arrival flows with multiple sources. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 399–410. IEEE, 2006.

[4] Robert Benkoczi, Bhattacharya Bhattacharya, Ante Ćustic, Sandip Das, and Tsunehiko Kameda. Facility location problems in cycle networks. *Japan conference on discrete and computational geometry, graphs, and games*, pages 139–140, 2017.

[5] Robert Benkoczi, Binay Bhattacharya, Yuya Higashikawa, Tsunehiko Kameda, and Naoki Katoh. Minsum k-sink problem on dynamic flow path networks. In *International Workshop on Combinatorial Algorithms*, pages 78–89. Springer, 2018.

[6] Robert Benkoczi and Rajib Das. Minmax sink location problem on dynamic flow cycle networks. *Submitted to CIAC-2019.*

[7] Binay Bhattacharya, Mordecai J Golin, Yuya Higashikawa, Tsunehiko Kameda, and Naoki Katoh. Improved algorithms for computing k-sink on dynamic flow path networks. In *Workshop on Algorithms and Data Structures*, pages 133–144. Springer, 2017.

[8] Di Chen and Mordecai J Golin. Minmax centered k-partitioning of trees and applications to sink evacuation with dynamic confluent flows. *arXiv preprint arXiv:1803.09289*, 2018.

[9] Siu-Wing Cheng, Yuya Higashikawa, Naoki Katoh, Guanqun Ni, Bing Su, and Yinfeng Xu. Minimax regret 1-sink location problems in dynamic path networks. In *International Conference on Theory and Applications of Models of Computation*, pages 121–132. Springer, 2013.

[10] Lester R Ford Jr and Delbert Ray Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958.

[11] Greg N Frederickson. Optimal algorithms for tree partitioning. In *SODA*, volume 91, pages 168–177, 1991.

[12] Greg N Frederickson and Donald B Johnson. Finding kth paths and p-centers by generating and searching good data structures. *Journal of Algorithms*, 4(1):61 – 80, 1983.

[13] Greg N Frederickson and Donald B Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal on computing*, 13(1):14–30, 1984.

[14] Horst W Hamacher and Stevanus A Tjandra. Mathematical modelling of evacuation problems: A state of art. *in: Pedestrian and Evacuation Dynamics, Springer Verlag*, pages 227–266.

[15] Yuya Higashikawa. *Studies on the Space Exploration and the Sink Location under Incomplete Information towards Applications to Evacuation Planning*. PhD thesis, Kyoto University, 2014.

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

[17] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[18] Satoko Mamada, Kazuhisa Makino, and Satoru Fujishige. Optimal sink location problem for dynamic flows in a tree network. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 85(5):1020–1025, 2002.

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

[20] Nimrod Megiddo. Combinatorial optimization with rational objective functions. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 1–12. ACM, 1978.

[21] Yinfeng Xu and Hongmei Li. Minimax regret 1-sink location problem in dynamic cycle networks. *Information Processing Letters*, 115(2):163–169, 2015.