ON THE CAPACITY PROVISIONING ON DYNAMIC NETWORKS

OLUWASEUN FRANCIS LIJOKA
Bachelor of Science, Federal University of Technology, Akure, Nigeria, 2009
Master of Science, Kwame Nkrumah University of Technology/AIMS-Ghana, Ghana, 2013
Ph.D. Mathematics, Heriot-Watt University United Kingdom, 2017

A thesis submitted
in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

THEORETICAL AND COMPUTATIONAL SCIENCE

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

ON THE CAPACITY PROVISIONING ON DYNAMIC NETWORKS

OLUWASEUN FRANCIS LIJOKA

Date of Defence:September 23, 2022

Dr. R. Benkoczi Thesis Supervisor	Professor	Ph.D.
Dr. H. Shahadat Thesis Examination Committee Member	Professor	Ph.D.
Dr. M. Khan Thesis Examination Committee Member	Assistant Professor	Ph.D.
Dr. M. Tata Internal Examiner Faculty of Arts and Science Department of Neuroscience	Professor	Ph.D.
Dr. M. Golin External Examiner Hong Kong University of Science and Technology, Hong Kong.	Professor	Ph.D.
Dr. W. Osborn Chair, Thesis Examination Committee	Associate Professor	Ph.D.

What we know is a drop, what we don’t know is an ocean

Sir Isaac Newton.

Dedication

I dedicate this project to God Almighty the creator of the universe and to my late mother Mrs Teniola Grace Lijoka.

Abstract

In this thesis, we consider the development of algorithms suitable for designing evacuation procedures in sparse or remote communities. The works are extensions of sink location problems on dynamic networks, which are motivated by real-life disaster events such as the Tohoku Japanese Tsunami, the Australian wildfire and many more. The available algorithms in this context consider the location of the sinks (safe-havens) with the assumptions that the evacuation by foot is possible, which is reasonable when immediate evacuation is needed in urban settings. However, for remote communities, emergency vehicles may need to be dispatched or situated strategically for an efficient evacuation process. With the assumption removed, our problems transform to the task of allocating capacities on the edges of dynamic networks given a budget capacity c. We first of all consider this problem on a dynamic path network of n vertices with the objective of minimizing the completion time (minmax criterion) given that the position of the sink is known. This leads to an $O(n\log n + n\log(c/\xi))$ time, where ξ is a refinement or precision parameter for an additional binary search in the worst case scenario. Next, we extend the problem to star topologies. The case where the sink is located at the middle of the star network follows the same approach for the path network. However, when the sink is located on a leaf node, the problem becomes more complicated when the number of links (edges) exceeds three. The second phase of this thesis focuses on allocating capacities on the edges of dynamic path networks with the objective of minimizing the total evacuation time (minsum criterion) given the position of the sink and the budget (fixed) capacity. In general, minsum problems are more difficult than minmax problems in the context of sink location problems. Due to few combinatorial properties discovered together with the possibility of changing objective

function configuration in the course of the optimization process, we consider the development of numerical procedure which involves the use of sequential quadratic programming (SQP). The sequential quadratic programming employed allows the specification of an arbitrary initial capacities and also helps in monitoring the changing configuration of the objective function. We propose to consider these problems on more complex topolgies such as trees and general graph in future.

Acknowledgments

I would like to acknowledge the contribution of individual that has made this work a success in one way or the other. The greatest share of my gratitude goes to my supervisor Professor Robert Benkoczi for being there always as a constant source of support, idea, courage and encouragement. I also appreciate him for his patience and for being a good teacher whom every student would dream to have. I can't thank him enough for teaching me how to think outside the box, analyse a problem, and eventually provide a solution to the problem. I also appreciate his assistance towards my family in general as we are lucky to have Mrs Benkoczi as our family doctor. I appreciate you a lot!

My next big appreciation goes to my committee members, Dr. Hossain Shahadat and Dr. Muhammad Khan for their contributions in discussion, positive criticism and ideas. I thank you both for always being there when I need to talk about my project and also during presentation. You will always be part of my life. My utmost gratitude also goes to Professor Wendy Osborn for teaching me advanced data analysis and for supporting all my applications for scholarships. You will always be part of my story. I will never forget to appreciate our admin support, Barb Hodgson for her gentleness and help within the department. I appreciate all the staff of math and computer science, including Professor Daya Gaou, Dr. Jana Archibald, and Dr. John Sheriff for their support. I will not forget to appreciate Leila Karimi and Sajad Fathi for organizing time for brainstorming and discussing ideas.

Next, I would like to appreciate my wife Margaret Ayoola for her support and understanding during the duration of the program. I really appreciate your drive and ambition to move the family forward. I cherish you a lot. I am also grateful to Bolakunmi Banjo and Lara Banjo for their support in taking care of my son Joshua and for always being there

when my family is in need of one help or the other. I am grateful to Olushoga Fashuyi for supporting me financially and the family of Mrs Toyin Dada and Mrs Soboyejo for granting us accommodation when we arrived in Lethbridge.

I gratuitously acknowledged the presence of some individuals that add fun to my life during the program. In particular, my son Oluwasinaayomi Joshua who arrived just at the beginning of my program. I would like to appreciate David Adebeshin, Fashuyi Ayodeji, Dr. Felix Ighagbon, and all members of Canadian Red Cross in Lethbridge. I also want to appreciate my cousin Dr. Akin Ojagbemi and my father inlaw Dr. Mathew Ayoola for their support and encouragement. Finally, I appreciate the God Almighty who is the source of all wisdom.

Contents

Dedication iv

Abstract v

Acknowledgments vii

List of Tables xi

List of Figures xii

1 Introduction and Background 1
- 1.1 Concept of the minmax criterion with a fixed sink . . . 3
- 1.2 Concept of minsum sink location problem with a fixed sink . . . 6

2 Background information and literature review 9
- 2.1 Overview of the sink location problem . . . 9
- 2.2 Review of data structure for minmax evacuation criterion . . . 14
  - 2.2.1 The critical and upper envelope tree structure . . . 15
- 2.3 Review of data structures for minsum evacuation criterion . . . 18
  - 2.3.1 The 1-sink minsum problem . . . 19
- 2.4 Numerical approach to non-linear constrained problems . . . 22
  - 2.4.1 The optimality conditions . . . 25
  - 2.4.2 Obtaining global optimum solution . . . 28
    - Convex programming . . . 29
  - 2.4.3 Unconstrained techniques for solving constrained problems . . . 30
    - Sequential unconstrained minimization techniques (SUMT) . . . 31
    - The multiplier methods . . . 32
- 2.5 The sequential quadratic programming (SQP) method . . . 33
  - 2.5.1 SQP notations and model formulation . . . 34
  - 2.5.2 Quadratic subproblem formulation . . . 36
  - 2.5.3 The SQP algorithm framework . . . 38
  - 2.5.4 Newton version of the SQP . . . 39
  - 2.5.5 Other variants of SQP methods . . . 41
  - 2.5.6 Full Hessian methods . . . 42
  - 2.5.7 The reduced Hessian method . . . 44
  - 2.5.8 Merit function for SQP . . . 46

3 Minmax capacity provisioning on path networks **50**
3.1 Model formulation and problem definition 51
3.2 Properties of the optimal solution for an arbitrary path with the sink located at an endpoint 64
3.3 Capacity provisioning for an arbitrary path network with the sink fixed at an endpoint 67
3.3.1 Implementation details and data structure 72
3.4 Capacity provisioning for evacuation of an arbitrary path with one arbitrary sink fixed 77

4 Capacity provisioning on star networks **79**
4.1 Star network with a sink in the middle 79
4.1.1 Algorithm for star networks with a sink in the middle 81
4.2 Star networks with more edges and a sink located at the centre 81
4.3 Three edged star network with the sink located on a leaf 84
4.3.1 Algorithm strategy 86

5 Minsum capacity provisioning on path networks **92**
5.1 Preliminaries and model formulation for the minsum capacity provisioning concept 93
5.2 Properties of the optimal solution vector and cost 103
5.3 Extension to longer path networks 105
5.4 Solution by sequential quadratic programming 110
5.4.1 The SQP model and algorithm formulation 111
5.4.2 The Hessian approximation 115
5.4.3 Numerical experiment 117

6 Conclusion and future work **120**

List of Tables

2.1 Table of minmax algorithms with their topologies and complexity in literature. [U] represents uniform edge capacities and [G] general edge capacities. 13
2.2 Table of minsum algorithms with their topologies and complexity in literature. [U] represents uniform edge capacities and [G] general edge capacities. 13

5.1 Table showing the solution for 2-edged network with fixed budgeted capacity $c = 20$ and 4 iterations. 117
5.2 Table showing the solution for 2-edged network with fixed budgeted capacity $c = 10$. We reduce the budgeted capacity to let the initial z indicate a congestion scenario. The optimal solution however indicate no congestion. The algorithm takes 4 iterations to complete. 117
5.3 Table showing the solution for 3-edge network with 5 iterations. 118
5.4 Table showing the solution for 4-edged network with $c = 20$. The algorithm takes 8 iterations to complete. 119
5.5 Solution for the case where the sink is in the middle of the path. The solutions are monotone from either sides. The algorithm takes 4 iterations to complete. 119

List of Figures

1.1 Two-edge network with a sink fixed at the right end. 3
1.2 A path network with a sink located on an edge. 5
1.3 Time-weight graph modeling the objective function for the two-edged path network of Figure 1.1 illustrating congestion free scenario. 7
1.4 Time-weight graph modeling the objective function for the two-edged path network of Figure 1.1 illustrating congestion scenario. 8

2.1 Three sinks located on a path network. The picture shows how subpaths are isolated with a sink fixed in each. 15
2.2 Balanced tree constructed on the vertices. 16

3.1 Simple path network with sink in the middle. 52
3.2 Illustration of capacities allocation on a two-edged path network with the sink located at the end vertex on the right. 55
3.3 Four-edged path network with sink at the middle. 59
3.4 Binary search procedure for the optimal capacity budget X_1. 62
3.5 Illustration for the proof of part (ii) from Theorem 3.7. 67
3.6 Example to illustrate longer path network with sink at the end. 67
3.7 Graph illustrating the optimal computation of x_2. 70
3.8 Explanation of how α_i is computed. We let $\alpha'_1, \ldots, \alpha'_t$ be the breakpoints of y_{i-1} and $p'_0, \ldots, p'_t$ the corresponding linear pieces. Only the line segment $[\alpha_i, \infty)$ is stored in the segment tree. 71
3.9 Seven arbitrary y'_i functions with their corresponding breakpoints. Only one-sided line segments $[\alpha_i, \infty]$ are stored in the segment tree. 74
3.10 Stored line segments $[\alpha_i, \infty)$ for y_i functions in Fig. 3.9. The vertical line shows an arbitrary query point $y_1 = \alpha'$ that cuts through only 3 line segments. 75
3.11 Illustration of the application of Algorithm 5 to an arbitrary path network with a fixed sink. 78

4.1 Simple star graph with sink in the middle. 80
4.2 Regular star network with more edges and sink at the centre 82
4.3 Three-edged tree network with a sink at its leaf 84
4.4 Graph illustrating the case when $x_0 \in (\frac{c}{2}, c)$ 87
4.5 Graph illustrating the case when $x_0 \in (\frac{c}{3}, \frac{c}{2})$. Note that the slope of f_{12} is not smaller than the larger of x_1 and x_2 but limited by x_0. 88
4.6 Graph showing the critical weight W_X at which $\theta_2 = \theta_{12}$. 90

5.1 Illustration of capacities allocation on a path network with two edges. The sink is assumed to be located at the end vertex on the right. 93

5.2 Time-weight graph of Figure 5.1. There is no congestion and evacuees get to the sink with their starting rates 94
5.3 Congestion scenario for the path network in Figure 5.1. The intersection of the two straight lines indicate the time at which $\theta_2(x_2,\eta) = \theta_1(x_1,\eta)$ 96
5.4 The case when $\eta = \eta_{\min}$ which corresponds to maximum value of x if the optimal solution is in congestion scenario 101
5.5 Path network with three edges and sink located on the right end. 105
5.6 Congestion free scenario where $\eta_{3,2} \leq w_3$ and $\eta_{2,1} \leq W[2,3]$ 106
5.7 Congestion only at v_3 107
5.8 Congestion only at v_2 108
5.9 Congestion at both vertices 109
5.10 Path network with two edges. 117
5.11 Path network with three edges. Fixed capacity at $c = 20$. 118
5.12 Path network with four edges. 118
5.13 Path network with four edges and a sink in the middle. 119

Chapter 1

Introduction and Background

The importance of evacuation planning or strategy is becoming increasing in our world today. Given the havoc and loss caused by recent natural disasters such as tsunami (in Tohoku city, Japan), earthquakes (in Haiti), flood (in parts of England) and wildfire (in Alberta, Canada) just to mention a few, governments of developed nations are particularly interested in efficient measures or strategies that could aid the quick evacuation of people, livestock and valuables immediately there is a sign or warning that demands such an evacuation.

The extent of destruction and loss of valuable properties and infrastructures may not be quantifiable after the disaster. Bridges may be swept away, mansions may be demolished, and train tracks wrecked just to mention a few. Above all, lack of an effective evacuation procedure may result in a large death toll which can add to the aftermath psychological trauma of the people and also cause a huge increase in the government disaster recovery cost. If a natural disaster is unavoidable, as the name indicates, huge cost of recovery and death toll are two negative impacts every government wants to avoid.

The required evacuation strategy in the event of emergency does not only involve time, but also the identification of safe locations within a street, region or the whole city at large. Depending on the scale of a disaster, evacuation strategy may also be planned within a school, hospital environment, engineering environment and any working environment.

In this thesis, we focus on developing algorithms suitable for the effective evacuation in remote or sparse communities. The existing algorithms in this context are developed with the assumption that evacuees can get to the sink by foot, which may not be feasible when

considering remote communities as the sink may be located more than hundred kilometers away. Therefore, emergency vehicles may have to be stationed strategically for an effective evacuation procedure. This leads to the problem of capacity provisioning on dynamic networks, given a budgeted (fixed) capacity and with the assumption that the sink position is known. Just as in traditional sink location problems, all algorithms considered in this work are classified into two optimal criteria. The first with more flurry of work is the minimax sink location which focuses on evacuation completion time while the second called the minsum sink location focuses on the sum of the evacuees' evacuation times.

The minmax and minsum sink location problems can be seen as the generalization of the NP−hard unweighted center and median problems respectively with respect to static networks [24]. In fact, if the edge capacities are large enough, both sink location problems reduce to the respective static problems. Extensive work up to tree networks have been considered for the case of the minimax criterion while only path networks have been investigated in the case of the minsum criterion in literature.

For the rest of this thesis, we shall use the following definitions and parameters. Let $P = (V,E)$ represent a path graph with a vertex set $V = \{v_1, v_2, \ldots, v_{n+1}\}$ and an edge set $E = \{e_i = (v_i, v_{i+1}), i = 1, \ldots, n\}$. Each vertex v_i has a non-negative weight w_i which represents the number of evacuees situated at v_i, and each edge e_i has a fixed length l_i but unknown capacity x_i, which represents the upper limit on the number of evacuees that can be evacuated through the edge e_i in a unit time. The parameter c denotes a given budgeted capacity to be shared optimally between all the edge lengths and τ denotes the transit time per unit distance.

We let $p \in P$ denote any point p that lies on the path graph P, and for $p,q \in P$, $p < q$ (resp. $p > q$) is interpreted as point p lies at the left (resp. right) of point q. We define $d(p,q)$ as the sum of edge lengths between p and q and if p and/or q are on an edge, we use prorated distance. We also denote the minimum capacity of the edges in a subpath connecting points p and q by $x(p,q)$. For evacuees to travel from p to q, $\tau d(p,q)$ time is

expended.

1.1 Concept of the minmax criterion with a fixed sink

Let us review the concept of the minmax evacuation criterion, which simply minimizes the evacuation maximum time or completion time in any topology of interest while also considering the effect of congestion that may ensue as a result of the edge capacities. The minmax problem has been studied up to tree topology and the approximation algorithm approach to a general graph network has been proposed recently in literature [35]. We review the concept by focusing on path networks with a fixed sink.

Problem 1.1. Given a path network P with a fixed sink (located anywhere on the path either a vertex or on an edge) and known number of evacuees w_i located at each vertex v_i. Let us assume that the edge length l_i and the edge capacity c_i for each edge $e_i = (v_i, v_{i+1})$, $i \geq 1$ are available. Let also assume that the transit time τ to travel a unit distance is known. The task is to minimize the maximum time of evacuation on the path network.

The above task is also equivalent to minimizing the evacuation time of the last evacuee on the network. Let us look at the following path network for example, where evacuees w_i and w_2 are located at vertices v_1 and v_2 respectively. The sink denoted by S is located (fixed) on the right end.

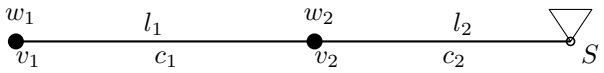

Figure 1.1: Two-edge network with a sink fixed at the right end.

Now, the time for the last evacuees at v_2 to get to the sink is

$$\tau l_2 + \frac{w_2}{c_2}, \tag{1.1}$$

where we refer to the first term as the distance time and the second term the waiting time. For the evacuees moving from v_1, we have to consider two cases:

Case 1 (No congestion): This means evacuees from v_1 are not impeded at v_2 as they journey to the sink. Mathematically this implies $\tau l_1 \geq \frac{w_2}{c_2}$. With this assumption, we have the evacuation time to be

$$T_{v_1} = \frac{w_1}{\min\{c_1, c_2\}} + \tau(l_1 + l_2), \tag{1.2}$$

where we note that that the waiting time to the sink is dictated by the minimum of the edge capacities.

Case 2 (Congestion): This is when evacuees from v_1 have to stop at v_2 due to backlog of evacuees waiting to evacuate through the edge e_2 to the sink. Mathematically this implies $\tau l_1 < \frac{w_2}{c_2}$. In this case the evacuation time is given by

$$T_{v_2} = \tau l_2 + \frac{w_1 + w_2}{c_2}. \tag{1.3}$$

Before we present a more interesting example, we introduce the following array

$$W[i] = W[1,i] = \sum_{j=1}^{i} w_j, \text{ for } i = 1,2,\ldots,n, \tag{1.4}$$

so that for $i \leq j$ we have

$$W[i,j] = W[j] - W[i-1]. \tag{1.5}$$

Similarly we define the prefix sum of the edge lengths

$$L[i] = L[1,i] = \sum_{j=1}^{i} l_j, \text{ for } i = 1,2,\ldots,n, \tag{1.6}$$

so that for $i \leq j$ we have

$$L[i,j] = L[j] - L[i-1]. \tag{1.7}$$

Also we let $c(i,j)$ denote the smallest capacity between subpath $P[v_i, v_j]$. Now let us consider the following path network where the sink is fixed in the interior of the path

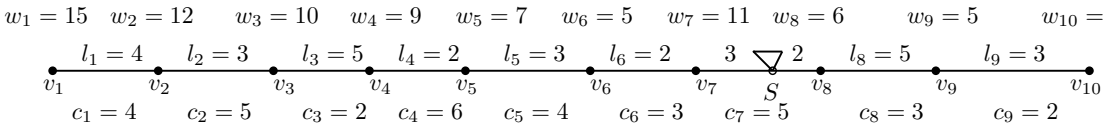

Figure 1.2: A path network with a sink located on an edge.

We have the left evacuation (where evacuees move from left to right) time with respect to a vertex $p \in [v_i, v_j]$ to the sink as

$$T_L^{[i,j]}(S,p) = \tau L[p,S] + \frac{W[i,p]}{c(p,S)}, \text{ for } S > j \tag{1.8}$$

and the right evacuation (where evacuees move from right to left) time is given as

$$T_R^{[i,j]}(S,p) = \tau L[S,p] + \frac{W[p,j]}{c(S,p)}, \text{ for } S < i. \tag{1.9}$$

As the sink is located on an edge say $e_{j'} = (v_{j'}, v_{j'+1})$ (resp. $e_{i'} = (v_{i'}, v_{i'+1})$), a prorated distance is used to compute the distance between the last vertex v'_j and S (resp. the distance between the last vertex v'_i and S). Let us compute the left evacuation time for the subpath $P[v_2, v_7]$ of the network above. This means we need to compute $T_L^{[2,7]}(S,p)$ for $p = 2,3,\ldots,7$. We have the following calculations based on equation (1.8)

$$T_L^{[2,7]}(S,2) = 18 + \frac{12}{2} = 24 \qquad T_L^{[2,7]}(S,3) = 15 + \frac{22}{2} = 26$$
$$T_L^{[2,7]}(S,4) = 10 + \frac{31}{3} = 20.33 \qquad T_L^{[2,7]}(S,5) = 8 + \frac{38}{3} = 20.67$$
$$T_L^{[2,7]}(S,6) = 5 + \frac{43}{3} = 19.33 \qquad T_L^{[2,7]}(S,7) = 3 + \frac{54}{5} = 13.8$$

From the above calculations, the maximum evacuation time for the subpath is 26 given by $T_L^{[2,7]}(S,3)$.

Definition 1.2. The vertex $v_p \in [v_i, v_j]$ that maximizes $T_L^{[i,j]}(S,p)$ is called the critical vertex [5].

For example, the critical vertex for the subpath $P[v_2, v_7]$ is v_3. The example above shows

that once the critical vertex within a subpath is determined, then the maximum evacuation time for the subpath can be quickly computed. With n being the number of vertices in the network, Bhattacharya *et al.* in [5] constructed a data structure called the capacities and upper envelope (CUE) tree to aid the determination of the critical vertices in $O(n\log^2 n)$ time while Higashikawa *et al.* in [24] employed a dynamic programming data structure to determine the critical vertex for the case when the edge capacities are uniform.

1.2 Concept of minsum sink location problem with a fixed sink

The minsum evacuation procedure focuses on minimizing the total evacuation time of all evacuees on a specified network. This criterion is more challenging than the minmax counterpart and all algorithms on the problem (in literature) are limited to path networks. Let us review the concept with respect to a fixed sink on a path network. Given a path network with a fixed sink and known number of evacuees w_i located at each vertex v_i. Suppose also that the edge length l_i and edge capacity c_i for each edge e_i are known. The task is to minimize the total evacuation time of all the evacuees to the sink. An important fundamental property known in literature for this problem is given in Theorem 1.3 below

Theorem 1.3. *The minsum sink must be on a vertex [24].*

Let us consider the two-edged path network in Figure 1.1. We have two cases

Case 1 (No congestion): We construct the following time-weight graph to illustrate the model or objective function for the case when $\tau l_1 \geq \frac{w_2}{c_2}$.

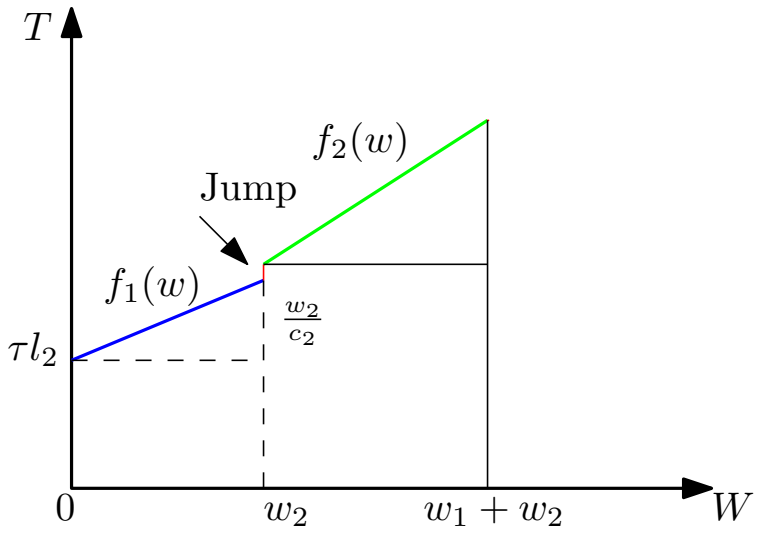

Figure 1.3: Time-weight graph modeling the objective function for the two-edged path network of Figure 1.1 illustrating congestion free scenario.

The $x-$axis is the prefix sum array of the weights while the $y-$axis is the time to the sink. For example, the first evacuees from v_2 get to the sink in time τl_2 while the rest have to wait $\frac{w_2}{c_2}$ unit of time before they proceed to the sink. The graph also shows a gap which corresponds to the time of zero flow-rate and no congestion at v_2. The areas of the two trapezoids in the time-weight graph gives the required evacuation total time.

$$z = w_2\tau l_2 + w_1\tau(l_1+l_2) + \frac{w_2^2}{2c_2} + \frac{w_1^2}{2\min(c_1,c_2)}. \tag{1.10}$$

Note also that we can derive the model by considering the time for the last evacuees at both v_1 and v_2 and computing the integral with respect to the evacuees. Thus we have

$$\begin{aligned} f_{v_2}(w) &= \tau l_2 + \frac{w}{c_2},\ w > 0 \\ f_{v_1}(w) &= \tau(l_1+l_2) + \frac{w-w_2}{\min(c_1,c_2)},\ w > w_2, \end{aligned} \tag{1.11}$$

and

$$\begin{aligned} z &= \int_0^{w_2} f_{v_2}\mathrm{dw} + \int_{w_2}^{w_1+w_2} f_{v_1}\mathrm{dw} \\ &= \tau l_2 w_2 + \tau(l_1+l_2)w_1 + \frac{w_2^2}{2c_2} + \frac{w_1^2}{2\min(c_1,c_2)}. \end{aligned} \tag{1.12}$$

Case 2 Congestion case: Let us assume that $\tau l_1 < \frac{w_2}{c_2}$, that is, evacuees from v_1 are delayed at v_2 due to some evacuees waiting to depart v_1 to the sink. The following time-weight graph illustrates the scenario:

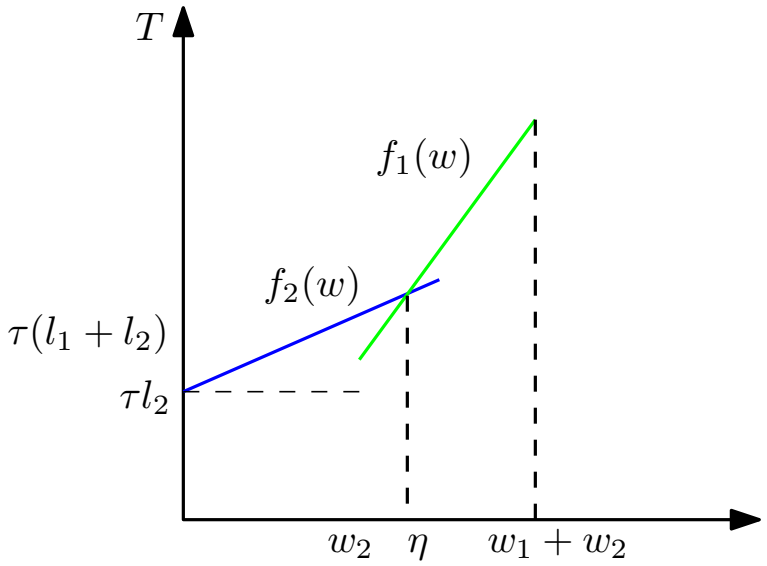

Figure 1.4: Time-weight graph modeling the objective function for the two-edged path network of Figure 1.1 illustrating congestion scenario.

Using the diagram above and Equation (1.11), we have the objective function to be

$$z = \frac{1}{2}(f_{v_2}(0) + f_{v_2}(\eta))\eta + \frac{1}{2}(f_{v_1}(\eta) + f_{v_2}(w_1+w_2))(w_1+w_2-\eta), \tag{1.13}$$

where η is the prefix sum weight which includes the evacuees from v_1 that get delayed at v_2. We can easily solved for η in the equation $f_{v_1}(\eta) = f_{v_2}(\eta)$.

Chapter 2

Background information and literature review

2.1 Overview of the sink location problem

The theory of network flow problems originated from a trivial notion of moving from one place to another together with the objective of reducing some cost function. This theory however opened a wide door for numerous interesting problems, where algorithm designers often wish to design procedures reliable to transport supplies from fixed sources to some target locations or sinks (not necessarily pre-established).

One of the well known applications of this theory is in the quickest transshipment problems. In this problem, a set of sources with specified amount of supplies and a set of sinks with pre-determined demands are given. The objective is to send exactly the right amount of supplies to the sinks in a shortest possible time. Ford and Fulkerson [29] in their quest to tackle this problem (for one source and sink) introduced the concept of dynamic flow network which has been exploited to tackle different evacuation problems. A dynamic network is a graph (path, tree or general graph) which models the movement of supplies (people, commodities, or other movable valuables) in a network. Each vertex in the graph holds some amount of supply and each edge is equipped with non-negative capacity, which limit the flow rate of supply through it. Also, the edges are often equipped with transit time which represents the time of travel in a unit distance.

Hoppe and Tardos in [10] investigated the problem further for multiple sources and sinks under the assumption of integral transit time. They presented the first polynomial

time algorithm for the problem. However, the complexity of their procedure is inapplicable in reality and as far as we know, finding a practical polynomial algorithm for the quickest transshipment problem is still open.

A related application of the dynamic flow theory is the evacuation problem (see survey paper [25]). In this problem, discrete supplies are given at some set sources and in contrast to the quickest transshipment problem, the demands for the set of sinks are not specified. The objective is to determine appropriate locations of the set of sinks in the network so as to minimize the travelling time of all evacuees on the network to these sinks. Two optimal criteria have been categorised in literature for the evacuation problem. The first known as the minmax criterion focuses on locating sinks on the network so as to minimize the evacuation completion time. Flurry of works have been published in this category even up to tree topology. The second category which focuses on the total evacuation time of all evacuees in the network is known as the minsum criterion. This category is harder than the former and as a result, few publications limited only to path networks have appeared in literature.

One important concept which characterized the flows in a network is the confluent or non-confluent property. Whenever there is a restriction that all flows entering a vertex must leave along same edge in the network, then we have confluent flows, otherwise non-confluent flows. Confluent flows are appreciated in evacuation problems for orderliness and to avoid confusion in the evacuation process. For both minmax and minsum optimal criteria, it is difficult to construct an optimal confluent flow when the underlying network is a general graph. In fact, if $P \neq NP$, a constant-factor approximate scheme for confluent flows cannot be achieved in polynomial time [39, 15]. The result of Hoppe and Tardos in [10] and PTAS result of Belmonte *et al* in [35] are based on non-confluent flow property, since there is no restriction on the flows in the network. Examples based on confluent flows can be seen in (evacuation) sink location problems where the underlying networks are trees and paths (see [12, 13, 24, 7] for examples.)

Under the minmax criterion with confluent flow, Mamada *et al* in [38] investigated the location of 1-sink on a dynamic tree network with general edge capacities by employing a table data structure. In particular, they constructed arriving and sending tables (at a specific destination vertex) which denote the arrival and departure rate as functions of time. The end of the arriving table clearly indicates the evacuation completion time. They assumed that the sink can only be on a vertex and developed an $O(n\log^2 n)$ time algorithm for the problem. Also on the tree network, Higashikawa *et al* in [23] and Bhattacharya and Kameda in [7] presented an $O(n\log n)$ time algorithms under the condition that the sink can be located on any part of the network. They exploited the unimodality of the objective function and the concept of tree's centroid in developing their algorithms. The most recent multiple sinks location algorithms under the tree topology was by Chen and Golin in [12] and [13]. They presented an $O(\max\{k,\log n\}\cdot kn\log^4 n)$ time algorithms with general edge capacities while investigating the location of k-sink in the network. Systematically, their algorithm isolates each subtree with enough sinks in a bottom up fashion until no subtree is left. Under the non-confluent characterization of flows with minmax criterion, Belmonte *et al* in [35] presented an approximation algorithm for locating sinks on general graph. Their algorithm employed the polynomial-time algorithm of Hoppe and Tardos [10] as subroutine to develop a fully polynomial time approximation scheme (FPTAS) [17, 32] for k-sink location on general network. They further showed that the problem is $W[1]$-hard [17, 32] when parametrized by k (the number of required sinks in the network). However, it is known that the complexity of their underlying approach is practically inapplicable.

Considering the path networks, Higashikawa *et al* in [24] presented an $O(kn)$ time algorithm under the condition that the edge capacities are equal and the flow is confluent. They formulated a recursive formulation for their approach and employed a 1-sink algorithm to locate a sink in a subpath divided by a $(k-1)$-dimensional divider. Improving on this was Bhattacharya *et al* in [5]. They presented $O(\min\{n\log^3 n, n\log n + k^2\log^4 n\})$ and $O(\min\{n\log n, n + k^2\log^2 n\})$ time algorithms for general and uniform edge capacities re-

spectively. Their algorithms are based on the Megiddo's parametric search [31] and sorted matrix method of Frederickson and Johnson [18]. They employed a novel tree data structure called the capacities and upper envelope tree (CUE tree) to facilitate their feasibility test as well as efficient methods to extract information from the tree data structure. The minmax k-sink location problem is a generalization of the NP-Hardness of the k-center problems in general graph [27, 20].

Regarding the minsum sink location problems, only the path algorithms exist so far in literature. The difficulty is due to the non-unimodal nature of the objective function. Higashikawa *et al* in [24] presented an $O(n)$ time algorithm for the case of one sink on a path network. They further showed that $k-$sink can be found in $O(\min\{kn^2, 2^{O(\sqrt{\log k \log\log n})}n^2\})$. Benkoczi *et al* in [37] presented improved algorithms for the minsum sink location on the same topology. They showed that $1-$sink minsum problem on path with general edge capacities can be solved in $O(n\log n)$ time complexity. They further derived $O(kn\log^3 n)$ and $O(kn\log^4 n)$ time algorithms for the $k-$sink location on path topology with uniform and general edge capacities respectively. These algorithms are based on efficient optimized dynamic programming formulation and a tree data structure called the cluster sequence tree which allowed efficient preprocessing of necessary information.

In comparison with the classical median problem, the minsum criterion can be reduced to the classical unweighted median problem by making the edge capacities as large as possible. Hence minsum optimal criterion is a generalization of the NP-hardness of the classical median problem [24, 37]. Tables 2.1 and 2.2 summarize the results known to date for the minmax and minsum sink location problems.

Topology	Problem	Time complexity
Path	$1-$sink [U]	$O(n)$ [24]
	$k-$sink [U]	$O(kn)$ [24], $O(n+k^2\log^2 n)$ [5], $O(n\log n)$ [7]
	$k-$sink [G]	$O(n\log n+k^2\log^4 n)$ [5], $O(n\log^3 n)$ [5]
Tree	$1-$sink [U]	$O(n\log n)$ [7, 23]
	$1-$sink [G]	$O(n\log^2 n)$ [38]
	$k-$sink [U]	$O(kn^2\log^4 n)$ [12], $O(\max\{k,\log n\}kn\log^3 n)$ [13]
	$k-$sink [G]	$O(kn^2\log^5 n)$ [12], $O(\max\{k,\log n\}kn\log^4 n)$ [12]
Cycle	$1-$sink [U]	$O(n)$ [4]
	$1-$sink [G]	$O(n\log n)$ [4]

Table 2.1: Table of minmax algorithms with their topologies and complexity in literature. [U] represents uniform edge capacities and $[G]$ general edge capacities.

Topology	Problem	Time complexity
Path	$1-$sink [U]	$O(n)$ [24]
	$k-$sink [U]	$O(n^2\cdot\min\{\sqrt{k\log n}+\log n, 2^{\sqrt{\log k\log\log n}}\})$ [24], $O(kn^2\log^3 n)$ [36]
	$k-$sink [U]	$O(kn\log^3 n)$ [37]
	$k-$sink [G]	$O(kn^2\log^2 n)$ [36], $O(kn\log^4 n)$ [37], $O(kn^2\log^2 n)$

Table 2.2: Table of minsum algorithms with their topologies and complexity in literature. [U] represents uniform edge capacities and $[G]$ general edge capacities.

All the novel algorithms discussed above emerged from the perspective of evacuating densely populated or urban communities. However, remote areas or sparsely populated communities, such as in British Columbia, Alberta, California and New South Wales are not left out from disasters, especially wildfires and floods. These areas are far from urban settings and evacuation by foot may not be efficient during unprecedented disasters, as

the safehaven maybe located very far away. Therefore in our research, we shall consider developing models and algorithms suitable for remote regions, where effective evacuation may not be possible by foot and emergency vehicles may have to be deployed or situated strategically to evacuate victims to safe haven (sink) hundreds of kilometers away.

2.2 Review of data structure for minmax evacuation criterion

The study of sink location problems present reliable procedures backed up by rigorous theoretical analysis for city planners to locate safe-havens in strategic positions and to aid the efficient evacuation process in the event of unforeseen disaster. In this section, we review an efficient algorithm proposed in [6] for locating k sinks on a path network, in such a way that the evacuation completion time is minimized. We acknowledge that Aramgutan *et al.* in [3] had presented an $O(kn\log^2 n)$ algorithm in 2014 while an $O(kn)$ algorithm was presented in [24] in 2015 for the case where the edge capacities are uniform, using a special dynamic programming data structure. However, the data structure in [6] proved more efficient as far as path network is concerned. As for tree networks, Mamada *et al* in [30] presented an $O(n\log^2 n)$ algorithm for 1-sink location with the assumption that the sink is located on a vertex, using a special table data structure, while an $O(n\log n)$ time algorithm for the case where the sink could be on any part of the tree was presented in [7].

With the important understanding of how crucial the critical vertices are to the computation of optimal cost (see definition 1.2) they construct an efficient data structure that can speed up the determination of these vertices, and also to aid the important task called the *feasibility test*. Given a time t, a problem instance is $t-$feasible if there exists a $k-$sink such that every evacuees can evacuate to a sink in time not exceeding t. The figure below shows how 3 sinks are located on a path network with 9 evacuee vertices.

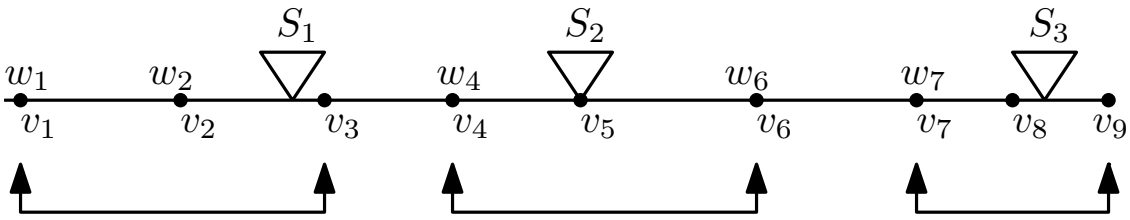

Figure 2.1: Three sinks located on a path network. The picture shows how subpaths are isolated with a sink fixed in each.

We generalize that $k < n$ otherwise, we can assign a sink on each vertex to have optimal cost of zero.

Let us describe the $t-$feasibility test before we explain the data structure. Starting with initial vertex v_a, find v_b such that all evacuees on the subpath $P[v_a, v_b]$ can be evacuated within time t but evacuees on $P[v_a, v_{b+1}]$ fail the test. This is called the left feasibility test and θ_L is used. Next locate a sink s on $(v_b, v_{b+1}]$ such that $\theta_L^{[a,b]}(v_b, C_L^{[a,b]}) + x\tau = t$ and $d(v_b, s) = x$. If $s \in (v_b, v_{b+1})$, set index $c = b+1$. If $s = v_{b+1}$ then set $c = b+2$. Then locate a vertex v_d such that evacuees on $P[v_c, v_d]$ can be evacuated to s within time t. This we call the right feasibility test and θ_R is used. The feasibility test is repeated and at each round, subpaths are isolated with a fixed sink until the last vertex v_n belongs to the last or kth isolated subpath.

2.2.1 The critical and upper envelope tree structure

To perform the $t-$feasibility test with respect to k sinks efficiently and to compute the critical vertices as quickly as possible, they construct a balance tree on the path network with the vertices being the leaves of the tree. Let v_j^+ be the right end of vertex v_j and let u_l and u_r be the left and right children of node u of the tree $\mathcal{T}$.

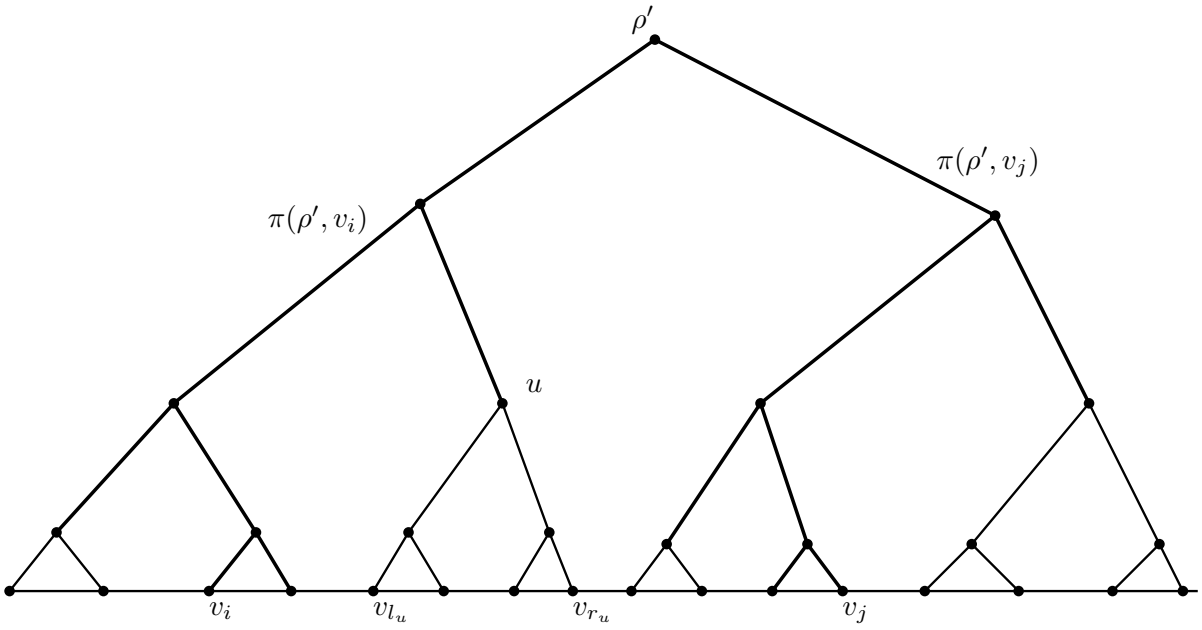

Figure 2.2: Balanced tree constructed on the vertices.

During preprocessing, the minimum capacities as well as the cumulative weights associated with any given subpath are stored at the nodes using a heap-like data structure bottom up in $O(n)$ time. We shall explain the left evacuation as the right evacuation procedure is symmetric. We recall that for any given subpath $P[v_i, v_j]$ the time to evacuate the evacuees within the subpath to a sink s beyond v_j is given by

$$\theta_L = \max_{v_p \in [v_i, v_j]} \left\{ d(v_p, v_j)\tau + W[v_i, v_p]/c(v_p, v_{j+1}) \right\} \tag{2.1}$$

Let us denote by $\pi(u, u')$ the path that connects the nodes u and u' of the tree $\mathcal{T}$ and let ρ' be the lowest common ancestor of any pair of leaves v_i, v_j. If we collect all the nodes that are the right children of the nodes on the path $\pi(v_i, \rho')$ and the left children of the nodes on the path $\pi(v_i, \rho')$ including v_i and v_j, (but ignoring the ones on the path) then we have a total of $O(\log n)$ such nodes. Let $N[v_i, v_j]$ be the set of the nodes and $\mathcal{P}(v_i, v_j)$ be the set of $O(\log n)$ subpaths spanned by the nodes in $N[v_i, v_j]$. With the existence of $N[v_i, v_j]$, formula

(2.1) is broken into

$$\max_{v_p \in V[v_{l_u}, v_{r_u}]} \left\{ d(v_p, v_{r_u})\tau + (W + W[v_{l_u}, v_p]) / \min\{c(v_p, v_{r_u+1}), c\} \right\}, \tag{2.2}$$

where $W = W[v_i, v_{l_u-1}]$ and $c = c(v_{r_u+1}, v_{j+1})$. Equation (2.2) is further broken down into two upper envelope functions with respect to each $u \in N[v_i, v_j]$. The first is a function of W given by

$$\begin{aligned} \theta^u_{L,1}(W) &= \max_{v_p \in V[v_{l_u}, v_{r_u}]} \left\{ d(v_p, v_{r(u)})\tau + (W + W[v_{l(u)}, v_p]) / c(v_p, v_{r(u)+1}) \right\} \\ &= \max_{v_p \in V[v_{l(u)}, v_{r(u)}]} \left\{ \theta_L^{[l(u), r(u)]}(v^+_{r(u)}, v_p) + \frac{W}{c(v_p, v_{r(u)+1})} \right\}, \end{aligned} \tag{2.3}$$

and the second is a function of c given by

$$\theta^u_{L,2}(c) = \max_{v_p \in V[v_{l(u)}, v_{r(u)}]} \left\{ d(v_p, v_{r(u)})\tau + \frac{W[v_{l(u)}, v_p]}{c} \right\}. \tag{2.4}$$

Both formulas are derived by considering the smaller of $c(v_p, v_{r(u)+1})$ and $c = c(v_{r(u)+1}, v_j)$. They are piecewise linear and continuous in their respective variables and therefore can be stored as sorted points using the idea of line duality and the computation of convex hull achievable in linear time by Graham scan algorithm.

Lemma 2.1. *Given a dynamic path network with n vertices, the capacity and upper envelope tree $\mathcal{T}$ can be constructed with associated data in $O(n \log n)$ in time and space.*

Once a pair of vertices (v_i, v_j) is given, the values of W and c are determined and the critical vertices can be determined by binary search over the bending points of $\theta^u_{L,1}(W)$ and $\theta^u_{L,2}(c)$, which is obtainable in $O(\log n)$ time. Thus, the $L-$critical vertices for $P[v_i, v_j]$ can be computed in $O(\log^2 n)$ because we need to compare $O(\log n)$ candidates of $N[v_i, v_j]$. With the CUE tree prepared, it takes $O(\log^3 n)$ time to locate a sink in an isolated subpath given a starting vertex v_a. This is because it takes $O(\log^2 n)$ to compute the critical

vertices as well as the left evacuation time and right evacuation time and then repeat the computation $O(\log n)$ time. Finally for general edge capacities, the $t-$feasibility test to insert k sinks on a path network can be carried out in $O(\min\{n\log^2 n, k\log^3 n\})$ time. The $O(k\log^3 n)$ complexity can be achieved by locating 1 sink in the first partition of the path network starting from v_1 in $O(\log^3 n)$ and then repeating the operation in $k-1$ more times. The $O(n\log^2 n)$ complexity is achievable by computing each left and right evacuation time sequentially which takes $O(n)$ and each computation takes $O(\log^2 n)$ time.

When the edge capacities are uniform, the critical vertices can be computed by concatenating two subpaths in constant time and for each node in $\mathcal{T}$ bottom up, which takes $O(n)$ time to construct the CUE tree. Therefore when the CUE tree is available, the left and right critical vertices for $P[v_i, v_j]$, $1 \leq i \leq j \leq n$ can be computed in $O(\log n)$ by exploring the $O(\log n)$ nodes in $N[v_i, v_j]$. Finally the $t-$feasibility test for the case of uniform edge can be carried out in $O(\min\{n, k\log n\})$ time provided the CUE tree is available.

2.3 Review of data structures for minsum evacuation criterion

We turn attention to review the problem of locating sinks on a path network in such a way that the evacuation total time is minimized. In particular, we review the 1-sink minsum work presented in [36] where special innovative method is employed, and for the $k > 1$ sinks the innovative method is combined with dynamic programming to solve the problem efficiently. We acknowledge that Higashikawa *et al* in [24] invented a dynamic programming approach for the case of uniform edge capacity network. They later converted the problem to an equivalent problem of searching for $k-$edge path in a complete weighted directed acyclic graph (DAG), an approach which yielded an $O(n^2 \cdot \min\{\sqrt{k\log n} + \log n, 2^{\sqrt{\log k \log\log n}}\})$ time. When $k = 1$, their algorithm runs in $O(n)$ time. The minsum sink location problem is more difficult than the minmax sink location on any topology and its studies so far in literature is limited to path topologies. One important property of the minsum problem which also differentiates it from the minmax is that the optimal location of the sink can only be on

a vertex.

With reference to a particular vertex v_i which could also be the location of the sink, if the arrival flow rate at v_i (or departure flow rate from v_i) is plotted against time, then we have a sequence of clusters. A cluster is regarded as the maximal time interval for which the arrival flow rate at v_i is continuously greater than zero. A cluster consists of sequence of sections which may occur as a result of congestion at the arriving vertex. Sections in a cluster have different heights, where the heights are given by the exiting edge capacities. We have a simple cluster when the cluster contains only 1 section, for example, when the edge capacity is uniform. The head vertex of a cluster is the vertex from which the evacuees corresponding to the start time of the cluster originates. The head vertex contains the first set of evacuees who are not impeded on their way to the reference vertex. Thus, the time for which the first set of evacuees from a cluster reaches a reference vertex v_i is called the offset of the cluster. Finally, the time interval of flow rate zero between adjacent clusters is termed a gap.

2.3.1 The 1-sink minsum problem

Let us consider the case when $k = 1$. We define the following time function (or cost)

$$\begin{aligned} \theta_L(x) &= \text{Cost at } x \text{ due to the vertices on } P[v_1, v_i] \text{ if } v_i < x \leq v_{i+1} \\ \theta_R(x) &= \text{Cost at } x \text{ due to the vertices on } P[v_{i+1}, v_n] \text{ if } v_i \leq x < v_{i+1}, \end{aligned} \tag{2.5}$$

so that

$$\theta(x) = \theta_L(x) + \theta_R(x). \tag{2.6}$$

A point $x \in V$ that minimizes $\theta(x)$ is called a minsum $1-$sink. Thus, the total cost can be computed by summing all the cost due to the vertices of all sections. Therefore, we need a model to compute a cost due to a section. Given a section of height c with offset t_0 and

duration δ_t, the cost of the section is given by

$$\lambda t_0 + \frac{\lambda^2}{2c}, \tag{2.7}$$

where $\lambda = c\delta_t$ is the weight carried by the section. The first term of equation (2.7) is called the extra cost, which is the time required for the evacuees carried by the section to move from the vertex head to their destination. The second term is referred to as the intra-cost, and it is the time required for all evacuees in the section to move from their origin vertices to the head vertex.

Now, to solve the $1-$sink minsum problem, we need an efficient way of detecting the arrival section sequence denoted by $\alpha_R(v_i)$ and departure section sequence $\beta_R(v_i)$ (both assumed to be moving to the left). Computing $\alpha_L(v_i)$ and $\beta_L(v_i)$ can be taken symmetrical.

We note that $\alpha_R(v_n) = \beta_R(v_1) = \Lambda$, where Λ is an empty sequence. This is because there is no right arrival at v_n and no right departure from v_1. We also note that if the height of a section arriving v_i is higher than the exiting edge capacity c_{i-1} of edge e_{i-1} then the evacuees carried by that section cannot depart through v_i, hence the duration of such section gets stretched. There is also a possibility that an arriving section encounter a congestion at v_i due to some backlog of evacuees waiting to depart v_i. This prompts the use of gap filling operation where the delayed evacuees are used to fill the gaps in $\alpha_R(v_i)$. A section with higher height than c_{i-1} is adjusted using the ceiling operation to stretch the duration of the section. However, if the height of an arriving section is less than c_{i-1}, the underutilized capacity is used to accommodate as many delayed evacuees as possible. The following observations are paramount to the algorithm:

(i) A stretched section may end in a gap

(ii) A section may shrink due to the stretched preceding it while its start time pushed to a later time.

From the above observation, we realise that the number of sections in $\beta_R(v_i)$ is at most one

more than that in $\alpha_R(v_i)$ for $i = n, n-1, \ldots$. Moreover, since the evacuees coming from further vertices would encounter smaller capacities, the heights of the sections in $\beta_R(v_i)$ due to the evacuees on subpath $P[v_i, v_n]$ are non-increasing with time.

With the above properties at hand, we let $\lambda(C)$ denote the number of evacuees carried by a cluster C and for mth cluster in $\alpha_R(v_i)$, the ratio $\lambda(C_m)/t_m$ is recorded where t_m is the time difference between the start of C_m and the succeeding section C_{m+1}. We term the ratio $\frac{\lambda(C_m)}{t_m}$ the critical capacity.

Now, clusters are merged depending on if there is an interaction (congestion) at the departure vertices, using the filling and ceiling (if incoming height is greater than outgoing height) operation. To also facilitate the computation of the sequences, max-heap $\mathcal{H}$ is introduced to store the critical capacities. This lead to the following lemma

Lemma 2.2. *The section sequences* $\{\alpha_R(v_i), \beta_R(v_i) | i = 1, \ldots, n\}$ *can be computed in* $O(n \log n)$ *time.*

For the efficient computation of the extra cost, we first of all create an array of weights $W[i] = \sum_{j=i}^{n} w_j$. Now using the formula

$$E_i = E_{i+1} + d_i W[i+1]\tau, \tag{2.8}$$

for $i = n-1, n-2, \ldots, 1$, we can compute the extra cost. In fact E_{i-1} can be computed from E_i in constant time. The weight array $W[i]$ can be precomputed in linear time.

Lemma 2.3. *The extra costs* $\{E_i | 1 \leq i \leq n\}$ *can be computed in* $O(n)$ *time.*

To compute the intra cost efficiently, we classify all sections of the same height in different clusters as sections of the same group or height group. Suppose the arrival section sequence $\alpha_R(v_i)$ consists of groups of sections, say $\mathcal{G}$ of which the heights of the groups are non-increasing from the first to the last in $\mathcal{G}$. Let G be a group of $\mathcal{G}$ with height $h(G)$ consisting of sections $S_1, S_2, \ldots S_g$ such that $\lambda(S_q)$ is the sum of the weights carried by a

section S_q, $1 \leq q \leq g$. Easily, we can compute the sum of the intra cost of the group G by

$$I(G) = \sum_{q=1}^{g} \lambda(S_q)^2 / 2h(G). \tag{2.9}$$

Then the total intra cost can be computed using

$$I(\mathcal{G}) = \sum_{G \in \mathcal{G}} I(G). \tag{2.10}$$

However, changes occur in G when sections in $\alpha_R(v_i)$ are merged due to smaller capacity c_{i-1} encountered along e_{i-1} and adjustments have to be done for accurate computation. To do this, we add the square weight of the new stretched section and the square weight of the section not totally swallowed, and we subtract the sum of the square weight of the totally or partially swallowed up sections. This computation takes $O(n)$ time because the swallowed up section takes no part in the updating time. Based on the explanation above, we have the following lemma

Lemma 2.4. *The intra cost* $\{I_i | 1 \leq i \leq n\}$ *can be computed in* $O(n \log n)$ *time,*

and all together we have the following theorem

Theorem 2.5. *A minsum* $1-$*sink in path network can be found in* $O(n \log n)$ *time.*

2.4 Numerical approach to non-linear constrained problems

The general model for a constrained optimization can be written as follow

$$\begin{aligned} & \min_{\mathbf{x} \in \mathrm{R}^n} \quad f(\mathbf{x}) \\ \mathbf{s.t} \quad & h_i(\mathbf{x_k}) = 0, \ i \in \mathcal{E} \\ & g_j(\mathbf{x}) \leq 0, \ j \in I, \end{aligned} \tag{2.11}$$

where $\mathcal{E}$ and I are sets of indices for equality and inequality constraints respectively. Let us denote by p the size of $\mathcal{E}$ and by m, the size of I. We define the feasible region as

the set of points that satisfies all the constraints, and we denote by $\mathbf{x}^\star$ the solution of the problem. The solution vector $\mathbf{x}^\star$ can be local or global. It is a global solution when $f(\mathbf{x}^\star)$ is the lowest function value when compared to the values at all other feasible points. A local solution vector is a set of points at which the objective function is smaller than the values of all other feasible nearby points or neighborhood. Many algorithms especially, for nonlinear problems often seek local solutions because the global solutions are often difficult to locate. However, when the functions are convex, local solution $\mathbf{x}^\star$ is also global.

Let us denote the feasible set by Ω given by

$$\Omega = \{\mathbf{x} \mid h_i(\mathbf{x} = 0), i \in \mathcal{E}; g_j(\mathbf{x}) \leq 0, j \in I\}, \tag{2.12}$$

thus equation (2.11) can be rewritten compactly as

$$\min_{\mathbf{x} \in \Omega} \quad f(\mathbf{x}). \tag{2.13}$$

Optimality conditions are of two types. The first is the necessary conditions which must be satisfied by any solution point (under certain assumptions) and the second is the sufficient conditions which guarantee that $\mathbf{x}^\star$ is in fact a solution. For unconstrained problems, the optimality conditions are straight forward. The necessary condition is that the gradient of the function at the optimal solution must be equal to zero, while the sufficient condition is that the second derivative at the optimal solution must be positive definite. A solution vector $\mathbf{x}^\star$ is a local solution of problem (2.13) if $\mathbf{x}^\star \in \Omega$ and there exists a neighborhood $\mathcal{N}$ of $\mathbf{x}^\star$ such that $f(\mathbf{x}) \geq f(\mathbf{x}^\star)$ for all $\mathbf{x} \in \mathcal{N} \cap \Omega$, $\mathbf{x} \neq \mathbf{x}^\star$. The solution $\mathbf{x}^\star$ is said to be an isolated local solution if $\mathbf{x}^\star \in \Omega$ and there exists $\mathcal{N}$ of $\mathbf{x}^\star$ such that $\mathbf{x}^\star$ is the only local solution in $\mathcal{N} \cap \Omega$.

One of the crucial properties for characterising solutions in constrained optimization problems is the smoothness of the objective function and the constraints. The presence of constraints in the problem definition often introduces many kinks and sharp edges and

that does not mean that the constraint functions that describe the feasible region are non-smooth [33]. This is because a non-smooth constraint can be broken down into set of smooth (piecewise) constraints. For example $\|\mathbf{x}\|_1 = |x_1| + |x_2| \leq 1$ can be broken down into $x_1 + x_2 \leq 1$, $-x_1 + x_2 \leq 1$, $x_1 - x_2 \leq 1$, and $-x_1 - x_2 \leq 1$. Also, a non-smooth objective function can be reformulated as a smooth constrained problem.

One idea of dealing with the inequality constraints in the problem definition (2.11) is transforming them to a set of equality constraints by adding a set of variables called the slack variables [2]. Since the inequality constraints in our problem definition is of the form "$\leq$", their values are either negative or zero at a feasible point. Therefore, the slack variables value must be either positive or zero at optimality. Thus, $g_j(\mathbf{x}) \leq 0$ can be written as $g_j(\mathbf{x}) + s_j = 0$, $s \geq 0$. The slack variables are then treated as part of the unknowns and their values are determined as part of the solution set.

Note that when s_j for a particular constraint is zero, then such constraint g_j is active, otherwise we have an inactive constraint. Note also that the size of the unknown variables increases by this idea. To avoid additional constraints $s_j \geq 0$, we can add the square of the slack variables to the constraint set to have $\mathbf{g}(\mathbf{x}) + \mathbf{s}^2 = 0$.

Before we present the necessary and sufficient conditions for constrained problem (2.11), we give the following definitions.

Definition 2.6. [33, 2, 8] The active set $\mathcal{A}(\mathbf{x})$ at any feasible $\mathbf{x}$ consists of the equality constraint indices from $\mathcal{E}$ together with the indices of the inequality constraints j for which $g_j(\mathbf{x}) = 0$. That is

$$\mathcal{A}(\mathbf{x}) = \mathcal{E} \cup \{j \in I \mid g_j(\mathbf{x}) = 0\}. \tag{2.14}$$

At a feasible point $\mathbf{x}$, the inequality constraints g_j is said to be active if $g_j(\mathbf{x}) = 0$ and inactive if otherwise.

Definition 2.7 (LICQ [33])**.** Given a point $\mathbf{x}$ and the active set $\mathcal{A}(x)$, the linear independent constraint qualification (LICQ) holds if the set of the gradient of active constraints

$\{\nabla g_j(x),\ j \in \mathcal{A}(x)\}$ is linearly independent. If LICQ holds, all active constraints gradients are nonzero.

Definition 2.8 (Regular points)**.** A point $\mathbf{x}^\star$ satisfying the constraint $\mathbf{h}(\mathbf{x}^\star) = \mathbf{0}$ is said to be a regular point of the feasible set if $f(\mathbf{x}^\star)$ is differentiable and gradient vectors of all constraints at $\mathbf{x}^\star$ are linearly independent.

Definition 2.9. Given a feasible point $\mathbf{x}$ and the active constraint set $\mathcal{A}(x)$, the set of linearized feasible directions $\mathcal{F}(\mathbf{x})$ contains the vectors $\mathbf{d}$ that satisfy

$$\mathcal{F}(\mathbf{x}) = \begin{cases} \mathbf{d}^T \nabla h_i(\mathbf{x}) = 0 \ \forall\ i = 1,\dots,p \\ \mathbf{d}^T \nabla g_j(\mathbf{x}) \leq 0 \ \forall\ j \in \mathcal{A}(\mathbf{x}) \cap I. \end{cases} \tag{2.15}$$

2.4.1 The optimality conditions

We review the necessary and sufficient conditions needed to be satisfied by $\mathbf{x}^\star$ to be a local solution of (2.11). The first order conditions are connected with the properties of the gradients of the objective (in relation to unconstrained problems) and constraint functions (extended to constrained problems). We have conditions in the following theorem.

Theorem 2.10 (First order necessary conditions [33, 2, 8])**.** *Let $\mathbf{x}^\star$ be a regular point of the feasible set that is a local minimum for problem* (2.11)*. Then there exists Lagrange multipliers $\mu^\star$ (a p-vector) and $\lambda^\star$ (an m-vector) such that the Lagrangian function is stationary with respect to x_k, $k = 1,\dots,n$, μ_i $i = 1,\dots,p$, λ_j $j = 1,\dots,m$ and s_j, $j = 1,\dots,m$ at point $\mathbf{x}^\star$.*

(i) The Lagrangian function for the problem is given by

$$\begin{aligned} \mathcal{L}(\mathbf{x},\mu,\lambda,\mathbf{s}) &= f(\mathbf{x}) + \sum_{i=1}^{p} \mu_i h_i(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j (g_j(\mathbf{x}) + s_j^2) \\ &= f(\mathbf{x}) + \mu^T \mathbf{h}(\mathbf{x}) + \lambda^T (\mathbf{g}(\mathbf{x}) + \mathbf{s}^2) \end{aligned} \tag{2.16}$$

(ii) Gradient conditions

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial x_k} = \frac{\partial f}{\partial x_k} + \sum_{i=1}^{p} \mu_i^\star \frac{\partial h_i}{\partial x_k} + \sum_{j=1}^{m} \mu_i^\star \frac{\partial g_j}{\partial x_k} = 0,\ k = 1,\ldots,n \\
&\frac{\partial \mathcal{L}}{\partial \mu_i} = 0 \Longrightarrow h_i(\mathbf{x}^\star) = 0,\ i = 1,\ldots,p \\
&\frac{\partial \mathcal{L}}{\partial \lambda_j} = 0 \Longrightarrow (g_i(\mathbf{x}^\star) + s_j^2) = 0,\ j = 1,\ldots,m
\end{aligned} \tag{2.17}$$

(iii) Feasiblity check for inequality constraints. That is $s_j^2 \geq 0$ or $g_j \leq 0$ for $j = 1,\ldots,m$.

(iv) Switching conditions or the complementary slackness conditions:

$$\frac{\partial \mathcal{L}}{\partial s_j} = 0 \Longrightarrow 2\lambda_j s_j = 0,\ j = 1,\ldots,m. \tag{2.18}$$

(v) Non-negativity of the Lagrange multiplier for the inequality constraints: $\lambda_j^\star \geq 0$ $j = 1,\ldots,m$.

(vi) Regularity check: that is the matrix

$$G = (\nabla h_1, \nabla h_2, \ldots, \nabla h_p, \nabla g_{j_1}, \nabla g_{j_2}, \ldots, \nabla g_{q_x}) \tag{2.19}$$

is linearly independent. The index q_x is the size of the set of active constraints associated with the inequality constraints. If this condition holds, then the Lagrange multipliers for the constraints are unique.

The switching conditions in the above theorem also tells us that if the inequality constraint $g_j(\mathbf{x}) \leq 0$ is active at the candidate minimum point $\mathbf{x}^\star$ then the corresponding Lagrange multiplier must be non-negative [2].

The first order conditions help us to understand the relationship between the derivatives of the objective function and the active constraints at the optimal solution $\mathbf{x}^\star$. When these conditions are satisfied, a move along any vector $\mathbf{d}$ from $\mathcal{F}(\mathbf{x})$ either makes $\mathbf{d}^T \nabla f(\mathbf{x}^\star) > 0$

or $\mathbf{d}^T \nabla f(\mathbf{x}^\star) = 0$. When we have $\mathbf{d}^T \nabla f(\mathbf{x}^\star) = 0$, the second order derivative information helps us to understand whether a move along this direction increases or decreases the objective function f.

To establish the second order conditions, we first impose a stronger smoothness assumption on f, $\mathbf{h}$ and $\mathbf{g}$. We require the functions to be twice continuously differentiable. Now, given $\mathcal{F}(\mathbf{x})$ and some $\mu^\star$ and $\lambda^\star$ that satisfy the KKT conditions, we define the critical cone as

$$\mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star) = \{\mathbf{d} \in \mathcal{F}(\mathbf{x}) \mid \nabla[h_i\, g_j]^T \mathbf{d} = 0,\ i = 1, \ldots, p \text{ and all } j \in \mathcal{A}(\mathbf{x}^\star) \cap I \text{ with } \lambda_j^\star > 0\}. \tag{2.20}$$

Theorem 2.11 (Second order necessary conditions [2])**.** *Suppose that* $\mathbf{x}^\star$ *is a local solution of* (2.11) *and that the LICQ holds. Let* $(\lambda^\star, \mu^\star)$ *be the Lagrangian multiplier vector for which the KKT conditions hold. Then*

$$\mathbf{d}^T \nabla^2 \mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star)\mathbf{d} \geq 0,\ \forall\, \mathbf{d} \in \mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star). \tag{2.21}$$

Theorem 2.12. *(Second order sufficient condition [2]) Suppose that for some feasible point* $\mathbf{x}^\star \in \mathrm{R}^n$ *there are Lagrange multiplier vectors* $\lambda^\star$ *and* $\mu^\star$ *such that the KKT conditions hold. Suppose also that* $\mathbf{d}^T \nabla^2 \mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star)\mathbf{d} > 0$ *for all* $\mathbf{d} \in \mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star)$, $\mathbf{d} \neq \mathbf{0}$. *Then* $\mathbf{x}^\star$ *is a strict local solution of problem* (2.11).

The second order conditions can be reformulated in a slightly weaker version by using two-sided projection of the Hessian of the Lagrangian $H\mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star)$ onto the subspaces that are related to $\mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star)$. When the strict complementary conditions and LICQ hold, we have

$$\mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star) = \mathrm{Null}[\nabla[\mathbf{h}\mathbf{x}^\star,\ \mathbf{g}(\mathbf{x}^\star)]^T]_{j \in \mathcal{A}(\mathbf{x}^\star)} = \mathrm{Null}(G(\mathbf{x}^\star)) \tag{2.22}$$

where G is defined in (2.19). Let Z be a full column rank whose column span the space

$\mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star)$, that is

$$\mathcal{C}(\mathbf{x}^\star, \mu^\star, \lambda^\star) = \{Zu \mid u \in \mathrm{R}^{|\mathcal{A}(\mathbf{x}^\star)|}\}, \tag{2.23}$$

then the necessary condition can be written as

$$u^T Z^T H\mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star) Zu \geq 0 \; \forall \; u, \tag{2.24}$$

or $Z^T H\mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star)Z$ is semi positive definite. Similarly, the second order sufficient theorem becomes $Z^T H\mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star)Z$ is positive definite. The matrix Z can be computed numerically using the QR-factorization of the active constraints gradient.

2.4.2 Obtaining global optimum solution

Global solution to a constrained non-linear optimization problem can be obtained in two ways [2, 33]:

(i) Exhaustive search method, which involves calculating all the local minimum points and computing the cost function at those points. The point with the least value of cost function is then taken as the global minimum. This approach can be employed if the cost function is closed and continuous on a bounded feasible set

(ii) Convexity: the KKT conditions are sufficient to compute the global minimum points if we can show that the optimization problem is convex.

We focus on the second method in this subsection.

Definition 2.13. A set $\mathcal{S}$ is said to be convex if two points p_1 and p_2 are any points in $\mathcal{S}$ and the line segment joining the two points is entirely in $\mathcal{S}$ [2, 33].

Mathematically for two points $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ the line segment joining them is given by

$$\mathbf{x} = \alpha\mathbf{x}^{(2)} + (1-\alpha)\mathbf{x}^{(1)}, \; 0 \leq \alpha \leq 1. \tag{2.25}$$

If the whole line segment (2.25) is in $\mathcal{S}$, then $\mathcal{S}$ is convex.

We can determine if a given function $f(\mathbf{x})$ is convex in relation to a convex set.

Definition 2.14. A function $f(x)$ is called convex on the convex set $\mathcal{S}$ if the graph of the function lie below the line joining any two points on the curve $f(x)$. For $n-$dimensional space, $f(\mathbf{x})$ defined on a convex set $\mathcal{S}$ is said to be convex if the following inequality is satisfied [2]

$$f(\alpha(\mathbf{x}^{(2)}+(1-\mathbf{x}^{(1)}))) \leq \alpha f(\mathbf{x}^{(2)})+(1-\alpha)f(\mathbf{x}^{(1)}),\ 0\leq\alpha\leq 1. \tag{2.26}$$

To check or determine the convexity of a function, equation (2.26) is not often used as it may require an infinite number of pairs of points. The following theorem is handy to use

Theorem 2.15. *A $n-$dimensional variable function $f(x_1,x_2,\ldots,x_n)$ defined on a convex set $\mathcal{S}$ is convex if and only if the Hessian matrix of the function is positive semi-definite or positive definite at all points in the set. The function f is strictly convex if the Hessian matrix is positive definite for all points. The Hessian matrix of f is given by [2, 33]*

$$Hf = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2}\cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots\cdots & \vdots \\ \frac{\partial^2 f}{\partial x_n^2} & \frac{\partial^2 f}{\partial x_n \partial x_2}\cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.27}$$

The above theorem is necessary and sufficient condition to show if a function is convex.

Convex programming

For the constrained optimization problem, if function $g_j(\mathbf{x})$ is convex, then the singleton set $g_j(\mathbf{x})\leq e_j$ (e_j a constant) is convex [2]. Now, if all g_j for $j=1,\ldots,m$ are convex, then the set $\{g_j(\mathbf{x})\leq e_j,\ j=1,\ldots,m\}$ is also convex, since the intersection of convex functions is convex.

Theorem 2.16. *Let $\mathcal{S}_s$ constitute the points that satisfy the constraint functions. That is*

$$\mathcal{S}_s = \{\mathbf{x} \mid h_i(\mathbf{x}) = 0,\ i = 1,\ldots,p\ \ g_j(\mathbf{x}) = 0,\ j = 1,\ldots,m\}, \tag{2.28}$$

then $\mathcal{S}_s$ is convex if functions set **g** *are convex and functions* **h** *are linear [2, 33].*

Feasible set $\mathcal{S}_s$ is also convex if the set is defined by linear equality or inequality functions. Now if the cost function is convex over the feasible set, we have a convex programming problem. The following theorem gives us the information needed for the local and global solution for convex programming problems

Theorem 2.17. *If $f(\mathbf{x}^\star)$ is a local minimum for a convex function $f(\mathbf{x})$ defined on a convex feasible set $\mathcal{S}$, then $f(\mathbf{x}^\star)$ is also a global minimum and $\mathbf{x}^\star$ is the global minimum vector [2, 33].*

2.4.3 Unconstrained techniques for solving constrained problems

The study of unconstrained non-linear optimization problems has opened a bank of techniques that can be exploited to solve constrained optimization problems. Many of the methods include the Newton method, which offers a second-order approximation and the quasi-Newton methods, which combine the attractive features of the steepest descent and Newton method. To employ the unconstrained methods to solve constrained problems, some sort of transformations need to be done. The objective function of the constrained problem is often wedded together with its associated constraint functions to form a composite function. Penalty parameters are often incorporated in the composite function formed, and are included to penalise the new objective function if the constraints are violated. The values of the penalty parameters are directly proportional to the extent of the violation. That is, the bigger the violation, the larger the penalty value [2, 33].

When the composite objective function is constructed successfully with some set of penalty parameters, it can be minimized using any of the unconstrained methods. At every

iteration, the penalty parameters are adjusted inline with some conditions and the composite objective function is again redefined and minimized. The process continues until desired solutions are obtained.

We review the transformation method under two categories [2, 33]:

(i) Sequential unconstrained minimization technique (SUMT)

(ii) Multiplier method.

Both transformation categories transform the problem to the following form [2]

$$\Phi(\mathbf{x};\mathbf{r}) = f(\mathbf{x}) + P(\mathbf{h}(\mathbf{x}), \mathbf{g}(\mathbf{x}), \mathbf{r}), \tag{2.29}$$

where $\mathbf{r}$ denotes a vector of penalty parameters and P is a real-valued function that imposes a penalty on the cost function with the help of $\mathbf{r}$. We discuss the two categories in turn.

Sequential unconstrained minimization techniques (SUMT)

Under this categories are the penalty function method and barrier function method. The penalty function method constructs the real-valued function P in (2.29) in such a way that a positive value of P is added to $f(\mathbf{x})$ whenever there is a violation of constraint. An example is the quadratic loss function given as [33]

$$\begin{aligned} P(\mathbf{h}(\mathbf{x}), \mathbf{g}(\mathbf{x}), r) &= r\{\sum_{i=1}^{p} [h_i(\mathbf{x})]^2 + \sum_{j=1}^{m} [g_i^+(\mathbf{x})]^2\} \\ g_i^+(\mathbf{x}) &= \max(0, g_i(\mathbf{x})), \end{aligned} \tag{2.30}$$

where r is a scalar penalty parameter. It is easy to observe that the function P is always positive in value. We also observe that $g_i^+(\mathbf{x}) = 0$ if the inequality constraint is active ($g_i(\mathbf{x}) = 0$) or inactive ($g_i(\mathbf{x}) < 0$) but positive if the inequality constraint is violated.

The advantages of the method include the ability to search through infeasible region and the possibility of assigning an arbitrary initial solutions for both the solutions and penalty

parameter. The drawbacks of the method include the fact that the iterative process may stop prematurely and the final solution may not be feasible if this happens.

The barrier function methods designed for inequality-constrained problems include [33, 2]

(i) The inverse barrier function

$$P(\mathbf{g}(\mathbf{x}), r) = \frac{1}{r} \sum_{j=1}^{m} \frac{-1}{g_i(\mathbf{x})} \tag{2.31}$$

(ii) The Log barrier function

$$P(\mathbf{g}(\mathbf{x}), r) = \frac{1}{r} \sum_{j=1}^{m} \log(-g_i(\mathbf{x})). \tag{2.32}$$

Both functions above give an infinite value whenever any of the inequality function is active. Therefore it is not advisable to start the method with some feasible solutions due to large barrier set around the feasible set. We can observe that as $r \to \infty$, $\mathbf{x}(r) \to \mathbf{x}^\star$. Some of the weaknesses of the barrier methods include instability (ill-behavior) near the boundary of the feasible set, unclear method of choosing or selecting the iterates of the penalty parameters and the ill-conditioning of the Hessian of the unconstrained function as $r \to \infty$

The multiplier methods

The methods under this category provide ways of circumventing the difficulties of the methods under the SUMT category. To make sure that the composite function Φ has a good conditioning, the transformation is carried out in such a way that the penalty parameters need not go to infinity. The methods also have the advantage of faster convergence rate compared to the SUMT methods with an arbitrary starting solution. The multiplier methods are also referred to as the augmented Lagrangian methods in literature. Below is an example of the method, whose construction introduces a penalty and multiplier for each constraint

[2]

$$P(\mathbf{h}(\mathbf{x}),\mathbf{g}(\mathbf{x}),\mathbf{r},\theta)=\frac{1}{2}\sum_{i=1}^{p}r_i'(h_i+\theta_i')^2+\frac{1}{2}\sum_{j=1}^{m}r_j((g_j+\theta_j)^+), \tag{2.33}$$

where $\theta_j>0$, $r_j>0$, and θ_i', $r_i'>0$ are parameters associated with the jth inequality and ith equality constraints respectively. Also, $g(_i+\theta_i)^+=\max(0,g_i+\theta_i)$. It is easy to observe that if $\theta_i'=\theta_j=0$ and $r_i'=r_j'=r$, then we have the quadratic loss function (2.30) which requires $r\to\infty$ in order to enforce convergence.

For a special case of equality-constrained problem, the augmented or composite function can be defined as

$$\Theta_E(\mathbf{x},\mathbf{h}(\mathbf{x}),r)=f(\mathbf{x})+\sum_{i=1}^{p}(\mu_i h_i(\mathbf{x})+\frac{1}{2}rh_i^2(\mathbf{x})), \tag{2.34}$$

where $r>0$ is a penalty parameter and μ_i is the Lagrange multiplier for the ith equality constraint. In general for equality-inequality constrained problem, the multiplier method can be written as [33]

$$\Phi(\mathbf{x},\mathbf{g}(\mathbf{x}),r)=\Phi_E(\mathbf{x},\mathbf{h}(\mathbf{x}),r)+\sum_{j=1}^{m}\begin{cases}\lambda_j g_j(\mathbf{x})+\frac{1}{2}rg_j^2(\mathbf{x}) & \text{if } g_j+\frac{\lambda_j}{r}\geq 0\\ -\frac{1}{2r}\lambda_j^2, & \text{if } g_j+\frac{\lambda_j}{r}<0,\end{cases} \tag{2.35}$$

where $\lambda_j\geq 0$ represents the Lagrange multiplier for the jth inequality constraint.

2.5 The sequential quadratic programming (SQP) method

The sequential quadratic programming (SQP) is arguably the most successful method for solving nonlinear constrained optimization problems [8, 2, 33]. It is regarded as a conceptual method from which different algorithms have emerged. The SQP was seemed to have been invented in 1963 by R.B Wilson in his PhD thesis titled "A simplicial algorithm for concave programming" [41], where he called the method Newton-SQP algorithm. However, the method became popular in the late 70s through the works of Garcia Palomares and

Mangasarian [34] and Han [21], where methods for unconstrained optimization problems were embedded in the SQP concept.

The SQP method basically models the nonlinear constrained problem for a given iterate $\mathbf{x}_k$, $k \in \mathrm{N}_0$ by a quadratic programming (QP) subproblem, and then uses the solution to the subproblem to construct a better iterate $\mathbf{x}_{k+1}$. Iterating this process leads to a sequence of iterates that is hoped will converge to a local solution $\mathbf{x}^\star$ as $k \to \infty$. The method shares some properties of Newton-type methods such as rapid convergence, especially when the iterates are close to the solution. It may behave erratically when the iterates are far from the solution, however some techniques have been invented to control this phenomenon [8].

The SQP method allows infeasible initial starting solution which is a big advantage when solving a nonlinear programming problem with nonlinear constraints. The method also depend largely on the existence of rapid and accurate algorithms and techniques for solving quadratic programming problems. Fortunately, there are many good procedures for solving quadratic programming problems.

2.5.1 SQP notations and model formulation

We briefly discuss here the required notations, process, and subquadratic model for the SQP method. We also list some assumptions needed for the method, which include the blanket assumption that all functions in the problem definition are three times continuously differentiable. We denote by $\nabla f(\mathbf{x})$ to be the gradient of a scalar function f, i.e., $\nabla f(\mathbf{x}) = [\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n}]^T$. We also employ the same notation ∇ to denote the Jacobian of a vector-valued function, for example, $\nabla \mathbf{h}(\mathbf{x}) = (\nabla h_1(\mathbf{x}), \nabla h_2(\mathbf{x}), \ldots, \nabla h_p(\mathbf{x}))$. We let $\mathcal{H}$ denote the Hessian of a scalar-valued function, which is a symmetric matrix with entries of the form $Hf(\mathbf{x})_{i,j} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$.

Paramount to the SQP method is the Lagrangian function

$$\mathcal{L}(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) + \mu^T \mathbf{h}(\mathbf{x}) + \lambda^T \mathbf{g}(\mathbf{x}), \tag{2.36}$$

where $\mu \in \mathrm{R}^p$ and $\lambda \in \mathrm{R}^m_+$ are the Lagrangian multiplier vectors for the equality and inequality constraints respectively. We define the index set of active constraints by

$$I(\mathbf{x}) = \{i : g_i(\mathbf{x}) = 0\}, \tag{2.37}$$

and we denote by $G(\mathbf{x}) \in \mathrm{R}^{n\times(p+q_x)}$ the matrix given by

$$G(\mathbf{x}) = (\nabla h_1(\mathbf{x}), \ldots, \nabla h_p(\mathbf{x}), \nabla g_{i_1}(\mathbf{x}), \ldots, \nabla g_{i_{q_x}}(\mathbf{x})), \tag{2.38}$$

where $q_x = |I(\mathbf{x})|$. We denote by $\mathbf{x}^\star \in \mathrm{R}^n$ the local minimum point of the problem and we write $H\mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star)$, where $\mu^\star \in \mathrm{R}^p$, $\lambda^\star \in \mathrm{R}^m$ are the optimal multiplier vectors. We assume the following conditions pertinent to the optimal solution (see [8, 2, 33]).

(i) The first order necessary optimality condition: That is, there exist Lagrangian multipliers $\mu^\star \in \mathrm{R}^p$ and $\lambda^\star \in \mathrm{R}^m_+$ such that

$$\nabla\mathcal{L}(\mathbf{x}^\star, \mu^\star, \lambda^\star) = \nabla f(\mathbf{x}^\star) + \nabla\mathbf{h}(\mathbf{x}^\star)\mu^\star + \nabla\mathbf{g}(\mathbf{x}^\star)\lambda^\star = 0. \tag{2.39}$$

This condition is also known as the KKT conditions. If a feasible point $\mathbf{x}$ satisfies the KKT conditions, we call the point a critical point of the problem. Critical point may not be a local minimum point.

(ii) The columns of $G(\mathbf{x}^\star)$ are linearly independent.

(iii) The strict complementary slackness holds at $\mathbf{x}^\star$, that is, $g_i(\mathbf{x}^\star)\lambda_i^\star = 0$, for $j = 1, \ldots, m$. But if $g_i(\mathbf{x}^\star) = 0$, $\lambda_i^\star > 0$.

(iv) The positive definite condition on $H\mathcal{L}^\star$. That is, the Hessian of the Lagrangian with respect to $\mathbf{x}$ is positive definite on the null space of $G(\mathbf{x})^T$. That is $\mathbf{d}^T H\mathcal{L}^\star\mathbf{d} > 0\ \forall$ $\mathbf{d} \neq 0$ such that $G(\mathbf{x}^\star)^T\mathbf{d} = 0$.

The last three conditions are referred to as the second-order sufficient optimality conditions which ensure that $\mathbf{x}^\star$ is an isolated local solution of the problem and that the multipliers are unique.

2.5.2 Quadratic subproblem formulation

The quadratic subproblem is constructed in a way that it reflects local attributes of the original problem. First, we express f as its local quadratic approximation

$$f(\mathbf{x}) \simeq f(\mathbf{x_k}) + \nabla f(\mathbf{x_k})\mathbf{d_x} + \frac{1}{2}\mathbf{d_x}^T B_K \mathbf{d_x}, \tag{2.40}$$

where $\mathbf{d_x} = \mathbf{x} - \mathbf{x_k}$ and the direct or natural definition of matrix B_k is the Hessian of f at iterate $\mathbf{x_k}$. The constraints are also expressed by their local linear approximations

$$\begin{aligned} \mathbf{g}(\mathbf{x}) &\simeq \mathbf{g}(\mathbf{x_k}) + \nabla \mathbf{g}(\mathbf{x_k})\mathbf{d_x} \\ \mathbf{h}(\mathbf{x}) &\simeq \mathbf{h}(\mathbf{x_K}) + \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x}. \end{aligned} \tag{2.41}$$

This leads to the following quadratic programming subproblem

$$\begin{aligned} \min_{\mathbf{d_x}} \quad & \nabla f(\mathbf{x_k})^T \mathbf{d_x} + \frac{1}{2}\mathbf{d_x}^T B_k \mathbf{d_x} \\ \mathbf{s.t} \quad & \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x} + \mathbf{h}(\mathbf{x_k}) = \mathbf{0} \\ & \nabla \mathbf{g}(\mathbf{x_k})\mathbf{d_x} + \mathbf{g}(\mathbf{x_k}) \leq \mathbf{0}. \end{aligned} \tag{2.42}$$

The above formulation is reasonable and useful if the constraints are linear. The presence of nonlinear constraints in many constrained optimization problems requires the use of a related approach that involves the Lagrangian as the objective function. The related approach is given below:

$$\begin{aligned} \min_{\mathbf{d_x}} \quad & \nabla \mathcal{L}(\mathbf{x_k}, \mu_k, \lambda_k)^T \mathbf{d_x} + \frac{1}{2}\mathbf{d_x}^T B_k \mathbf{d_x} \\ \mathbf{s.t} \quad & \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x} + \mathbf{h}(\mathbf{x_k}) = \mathbf{0} \\ & \nabla \mathbf{g}(\mathbf{x_k})\mathbf{d_x} + \mathbf{g}(\mathbf{x_k}) \leq \mathbf{0}. \end{aligned} \tag{2.43}$$

The subproblems Eq.(2.42) and (2.43) are equivalent in some instances. First, if there is no inequality constraints. This is because the term $\nabla \mathbf{h}(\mathbf{x_k})^T \mathbf{d_x}$ is a constant and the functional in (2.43) reduces to that in (2.42). Secondly, if the multiplier λ_k in (2.43) satisfies $\lambda_k^i = 0$, for $i \in I_{in}(\mathbf{x_k})$. This is because only the active inequality constraints are important and since terms $\nabla \mathbf{h}(\mathbf{x_k})^T \mathbf{d_x}$ and $\nabla \mathbf{g}(\mathbf{x_k})\mathbf{d_x}$, $i \in I_{ac}(\mathbf{x_k})$ are constants, we have equivalency.

Another idea of formulating the subproblem suggests the addition of slack variables to the inequality constraints before constructing the subproblem. This idea leads to the following

$$\begin{aligned} \min_{\mathbf{d_x},\mathbf{d_s}} \quad & \nabla f(\mathbf{x_k})^T \mathbf{d_x} + \frac{1}{2}\mathbf{d_x}^T B_k \mathbf{d_x} \\ \mathbf{s.t} \quad & \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x} + \mathbf{h}(\mathbf{x_k}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x_k}) + s_k + \nabla \mathbf{g}(\mathbf{x_k})^T \mathbf{d_x} + d_s = \mathbf{0}, \end{aligned} \tag{2.44}$$

where $\mathbf{s} = \mathbf{t^2}$. The formulation is equivalent to the one below where the Lagrangian is used as the objective functional:

$$\begin{aligned} \min_{\mathbf{d_x},\mathbf{d_s}} \quad & \nabla \mathcal{L}(\mathbf{x_k}, \mu_k, \lambda_k)^T \mathbf{d_x} + \frac{1}{2}\mathbf{d_x}^T B_k \mathbf{d_x} \\ \mathbf{s.t} \quad & \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x} + \mathbf{h}(\mathbf{x_k}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x_k}) + s_k + \nabla \mathbf{g}(\mathbf{x_k})^T \mathbf{d_x} + d_s = \mathbf{0}. \end{aligned} \tag{2.45}$$

The quadratic subproblem is solved at each iteration and the solution together with the multipliers is updated using:

$$\begin{aligned} \mathbf{x_{k+1}} &= \mathbf{x_k} + \alpha \mathbf{d_x} \\ \mu_{\mathbf{k+1}} &= \mu_{\mathbf{k}} + \alpha \mathbf{d}_\mu \\ \lambda_{\mathbf{k+1}} &= \lambda_{\mathbf{k}} + \alpha \mathbf{d}_\lambda, \end{aligned} \tag{2.46}$$

where α is the steplength and

$$\begin{aligned} d_\mu &= \mu - \mu_k \\ d_\lambda &= \lambda - \mathbf{d}_\lambda. \end{aligned} \tag{2.47}$$

2.5.3 The SQP algorithm framework

Consistency of the quadratic subproblem is very crucial to all SQP algorithm variants. This implies that the associated system of constraints must have a non-empty feasible set. This requirement is guaranteed when $\mathbf{x_k}$ is close to $\mathbf{x}^\star$ supported also by condition imposed on matrix $G(\mathbf{x})$. However, some difficult problems may fail consistency condition even when $\mathbf{x_k}$ is in the neighborhood of $\mathbf{x}^\star$.

Another issue we need to consider is the convergence of the iterates. Iterates may converge locally or globally. Local convergence is based on the assumptions that the initial solution is relatively close to a local solution $\mathbf{x}^\star$ and the initial Hessian is close to $H\mathcal{L}^\star$. Apart from the non-singular requirement of $H\mathcal{L}^\star$, the following are the conditions imposed on $H\mathcal{L}^\star$ and B_k and are important for local convergence (see [33, 2, 8]):

(i) Uniform positive definiteness of B_k on the null space of $\nabla\mathbf{h}(\mathbf{x_k})^T$ is required. That is, there exists $\beta_1 > 0$ such that $\forall k$, $\mathbf{d}^T B_k \mathbf{d} \geq \beta_1 \|\mathbf{d}\|^2 \; \forall \mathbf{d}$ such that $\nabla\mathbf{h}(\mathbf{x_k})^T\mathbf{d} = \mathbf{0}$.

(ii) The sequence of matrix $\{B_k\}$ must be uniformly bounded. That is, there exists $\beta_2 > 0$, such that $\|\beta_k\| \leq \beta_2$

(iii) The inverses of matrices B_k must be uniformly bounded. That is there exists $\beta_3 > 0$ such that $\forall k$, B_k^{-1} exists and $\|B_k^{-1}\| \leq \beta_3$.

For global convergence, we assume that that initial iterate may be remote and a function that measures the progress of convergence is often incorporated. This function is called merit function denoted by Φ. It reduction at each step indicates a progress towards a solution. We present the SQP skeleton algorithm below [8, 2]:

Algorithm 1: SQP algorithm framework.

Result: Output the approximate solution vector for a given nonlinear constrained problem

(1) **Initialize** $(\mathbf{x}_0, \mu_0, \lambda_0)$ and B_0 ;

(2) **Choose** a merit function Φ and set $k = 0$;

(3) **Form** the QP and solve for $(\mathbf{d}_x, \mathbf{d}_\mu, \mathbf{d}_\lambda)$;

(4) **Choose** steplength α such that $\Phi(\mathbf{x}_k + \alpha\mathbf{d}_x) < \Phi(\mathbf{x}_k)$;

(5) **Set**

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{d}_x$$
$$\mu_{k+1} = \mu_k + \alpha\mathbf{d}_\mu$$
$$\lambda_{k+1} = \lambda_k + \alpha\mathbf{d}_\lambda$$

;

(6) **Stop** if converged ;

(7) **Otherwise** compute B_{k+1}, set $k = k+1$ and return to 3.

Since the full constraint nonlinear problem can be converted to equality-constrained problem by adding slack variables, we shall focus on the following equality-constrained quadratic programming for easy explanation of different variants of SQP method

$$\begin{aligned} \min_{\mathbf{d_x}} \quad & \nabla f(\mathbf{x_k})^T \mathbf{d_x} + \frac{1}{2}\mathbf{d_x}^T B_k \mathbf{d_x} \\ \mathbf{s.t} \quad & \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x} + \mathbf{h}(\mathbf{x_k}) = \mathbf{0}. \end{aligned} \tag{2.48}$$

2.5.4 Newton version of the SQP

The Newton-type SQP provides an algortithm with a unit steplength [8, 2]. The local convergence proof of SQP is also based on the Newton version. From the first order

necessary conditions apply to (2.48), we have

$$\begin{aligned} \nabla \mathbf{h}(\mathbf{x_k})\mu_{k+1} + B_k \mathbf{d_x} &= -\nabla f(\mathbf{x_k}) \\ \nabla \mathbf{h}(\mathbf{x_k})^T \mathbf{d_x} &= -\mathbf{h}(\mathbf{x_k}), \end{aligned} \tag{2.49}$$

where μ_{k+1} is a vector of multipliers at $k+1$. Setting $\mu_{k+1} = \mu_k + \mathbf{d}_\mu$, we have

$$\begin{aligned} B_k \mathbf{d_x} + \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d}_\mu &= -\nabla \mathcal{L}(\mathbf{x_k}, \mu_k) \\ \nabla \mathbf{h}(\mathbf{x_k})^T \mathbf{d_x} &= -\mathbf{h}(\mathbf{x_k}). \end{aligned} \tag{2.50}$$

Now, setting $B_k = H\mathcal{L}(\mathbf{x_k}, \mu_k)$ and applying Newton's method together with the first order necessary conditions, we have the right-hand side vector as

$$\psi(\mathbf{x}, \mu) = \begin{bmatrix} \nabla \mathcal{L}(x, \mu) \\ \mathbf{h}(\mathbf{x}) \end{bmatrix} = \mathbf{0}, \tag{2.51}$$

and the Jacobian matrix at the solution is

$$\mathcal{J}(\mathbf{x}^\star, \mu^\star) = \begin{bmatrix} H\mathcal{L}^\star & \nabla \mathbf{h}(\mathbf{x}^\star) \\ \nabla \mathbf{h}(\mathbf{x}^\star)^T & \mathbf{0}. \end{bmatrix} \tag{2.52}$$

The matrix $\mathcal{J}(\mathbf{x}^\star, \mu^\star)$ is assumed to be non-singular. The Newton SQP scheme now requires to solve

$$\mathcal{J}(\mathbf{x_k}, \mu_k)\mathbf{d}_{\mathbf{x},\mu} = -\psi(\mathbf{x_k}, \mu_k), \tag{2.53}$$

where $\mathbf{d}_{\mathbf{x},\mu} = (\mathbf{d_x}, \mathbf{d}_\mu)$ and the scheme is updated with a unit steplength. The initial iterate for the multiplier can be computed using (see [8])

$$\mathbf{u_0} = -[\nabla \mathbf{h}(\mathbf{x_0})^T \nabla \mathbf{h}(\mathbf{x_0})]^{-1} \nabla \mathbf{h}(\mathbf{x_0})^T \nabla f(\mathbf{x_0}). \tag{2.54}$$

Formula (2.54) follows the first order necessary condition and also referred to as the least square estimate [2, 8].

The advantages of the Newton-SQP scheme include the benefit of a unit steplength and rapid convergence with no need of line searches. It has been proven that if the initial iterate $(\mathbf{x_0}, \mu_0)$ is sufficiently close to $(\mathbf{x}^\star, \mu^\star)$, the scheme converges quadratically. However, it is practically difficult to choose an initial iterate that will be closer to the true local solution. Moreover, when the initial iterate is far from the true solution, $H\mathcal{L}(\mathbf{x_k}, \mu_k)$ cannot be assumed to be positive definite on the required subspace and this may not guarantee the existence of local solution to (2.48).

2.5.5 Other variants of SQP methods

The updating scheme for the matrix B_k play an important role in SQP algorithms. As we have noticed earlier, the Newton version uses a direct Hessian approximation as the matrix B_k, which of course has some practical advantages. Other variants of SQP method emerge from finding good approximation schemes for B_k. These schemes, however emerge from the in-depth study of unconstrained optimization problems. The variants are divided into two categories. The first called the full Hessian approximations approximate the matrix B_k to be close to $H\mathcal{L}$ as possible while the second referred to as reduced Hessian approximations compute matrices that approximate the Hessian on the null space of the Jacobian of the constraints [8]. One important property of the sequence $\{B_k\}$ necessary for the approximation of $H\mathcal{L}^\star$ is the bounded deterioration property defined below:

Definition 2.18. [8, 9] A sequence of matrix approximation, B_k, for the SQP method is said to have the bounded deterioration property if there exist constants α_1 and α_2 independent of k such that

$$\|B_{k+1} - H\mathcal{L}^\star\| \leq (1 + \alpha_1\alpha_k)\|B_k - H\mathcal{L}^\star\| + \alpha_2\alpha_k, \tag{2.55}$$

where

$$\alpha_k = \max\{\|\mathbf{x_k} - \mathbf{x_k}\|, \|\mathbf{x_k} - \mathbf{x}^\star\|, \|\mu_{k+1} - \mu^\star\|, \|\mu_k - \mu^\star\|\}.$$

We discuss the full Hessian approach first, followed by the reduced Hessian method.

2.5.6 Full Hessian methods

Under the full Hessian approximation methods, we have (see [8])

(i) **The rank-two Powell-Symmetric-Broyden (PSB) [21]**. This method approximate B_k using

$$B_{k+1} = B_k + \frac{1}{(S^T S)}(y - B_k S)S^T + S(y - B_k S)^T - \frac{(y - B_k S)^T S}{(S^T S)^2}(SS^T), \tag{2.56}$$

where

$$S = \mathbf{x_{k+1}} - \mathbf{x_k} \tag{2.57}$$

and

$$y = \nabla \mathcal{L}(\mathbf{x_{k+1}}, \mu_{k+1}) - \nabla \mathcal{L}(\mathbf{x_k}, \mu_{k+1}). \tag{2.58}$$

The sequence of matrices $\{B_k\}, k > 0$ may not be positive definite. It has been verified that if $\mathbf{x}_0$ is close to $\mathbf{x}^\star$, and B_0 is close to $H\mathcal{L}^\star$, then $\{B_k\}$ satisfies the bounded deterioration property and the iterates converges superlinear to a local solution. If the initial solution is chosen remotely, the scheme may suffer the same drawback of the Newton SQP, due to the positive indefiniteness of the sequence $\{B_k\}$. Moreover, global convergence scheme maybe difficult to establish.

(ii) **The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method**. The BFGS scheme focuses on generating sequence of B_k approximations that are positive definite. The formula is given by (see [8, 2, 33])

$$B_{k+1} = B_k + \frac{yy^T}{S^T y} - \frac{B_k(SS^T)B_k}{S^T B_k S}, \tag{2.59}$$

and

$$S^T y > 0. \tag{2.60}$$

The formula possess the bounded deterioration property and if B_k is positive definite and $y^T S > 0$, then B_{k+1} is positive definite. The main drawback is that condition (2.60) may not be satisfied. The advantages of the scheme are attractive. First, initial matrix B_0 can be chosen to be an identity matrix. Secondly, if the problem is convex, then it is easy to satisfy the positive definite requirement of $H\mathcal{L}^\star$. Also, due to the positive definite of B_k, the existence of solution to (2.48) is guaranteed even when the initial iterates are chosen remotely. Lastly, BFGS scheme is also backed with rigorous analysis in the realm of unconstrained optimization problems.

(iii) **The Powell-SQP method**. This scheme modifies the formula (2.59) by replacing y by

$$\hat{y} = \theta y + (1-\theta)B_k S, \text{ for } \quad \theta \in (0,1]. \tag{2.61}$$

The modification still preserves the positive definiteness of the Hessian approximation and the condition (2.60) is always satisfied. However, the secant condition satisfied by (2.59) no longer apply. Although the local convergence proof of the method is yet to be established, the method has performed experimentally very well.

(iv) **The augmented Lagrangian method**. This method transforms the objective function of the problem to

$$f_A(\mathbf{x}) = f(\mathbf{x}) + \frac{\eta}{2}\|h(\mathbf{x})\|^2, \ \eta > 0, \tag{2.62}$$

whose Lagrangian function is given by

$$\mathcal{L}_A(\mathbf{x},\mu) = \mathcal{L}(\mathbf{x},\mu) + \frac{\eta}{2}\|\mathbf{h}(\mathbf{x})\|^2, \tag{2.63}$$

and Hessian at $(\mathbf{x}^\star,\mu^\star)$ is

$$H\mathcal{L}_A(\mathbf{x},\mu) = H\mathcal{L}^\star + \eta\nabla\mathbf{h}(\mathbf{x}^\star)\nabla\mathbf{h}(\mathbf{x}^\star)^T. \tag{2.64}$$

BFGS method is then applied to the transformed objective function. The drawback of the problem is the difficulty in choosing η. In some problems, large value of η may be required to ensure the positive definiteness of $H\mathcal{L}^\star$, which also requires the a priori knowledge of $\mathbf{x}^\star$. Numerical instability may ensue if an arbitrary large η value is used.

(v) **The SALSA SQP method**. To circumvent the difficulties of the argumented Lagrangian SQP idea, the SALSA SQP method was invented and formulated in such a way that independent of $\mathbf{x}^\star$, a precise estimate of η can be chosen. Let

$$y_A = \nabla \mathcal{L}_A(\mathbf{x}_{k+1}, \mu_{k+1}) - \nabla \mathcal{L}_A(\mathbf{x_k}, \mu_{k+1}), \tag{2.65}$$

then we can write

$$y_A = y + \eta \nabla \mathbf{h}(\mathbf{x_{k+1}}) \mathbf{h}(\mathbf{x_{k+1}}), \tag{2.66}$$

where y is given in (2.58). Positive definiteness of the augmented BFGS scheme is preserved if the condition (2.60) is satisfied. The attractive feature also include the idea that a minimum value of η_k independent on $\mathbf{x}^\star$ can be given to ensure $y_A^T S > 0$ for values of $\mathbf{x}_k$, μ_{k+1} close to $(\mathbf{x}^\star, \mu^\star)$. The drawback include the possibility that $H\mathcal{L}^\star$ could fail positive definite test despite the possibility of positive definite of the sequence $\{B_k\}$. The bounded deterioration property may not be satisfied. Lastly, the local convergence theory is not yet established.

2.5.7 The reduced Hessian method

The reduced Hessian approximation of sequence of matrices $\{B_k\}$ is based on the condition that $H\mathcal{L}^\star$ is required to be positive definite on the null space of the gradient matrix $G(\mathbf{x})^T$ defined in (2.38) [8, 11]. Thus the methods approximate only the portion of the Hessian matrix pertinent to the subspace $G(\mathbf{x})^T$. Attractive advantages of the method are the size of the Jacobian matrix is reduced to $n-m$ and standard positive definite updating

scheme can be employed. We shall discuss the method in [8, 11].

Let $\nabla \mathbf{h}(\mathbf{x_k})$ estimated at x_k be a full rank matrix. Let us denote by Z_k and Y_k the matrices whose columns form bases for the null space of $\nabla \mathbf{h}(\mathbf{x})^T$ and range space of $\nabla \mathbf{h}(\mathbf{x_k})$ respectively. Matrices Z_k and Y_k can be computed by a QR factorization of $\nabla \mathbf{h}(\mathbf{x_k})$. We also assume that the columns of Z_k are orthogonal. The definition of a reduced Hessian is given below:

Definition 2.19. Let $(\mathbf{x_k}, \mu_k)$ be a given iterate and $\nabla \mathbf{h}(\mathbf{x_k})$ has a full rank. The matrix $Z_k^T H \mathcal{L}(\mathbf{x_k}, \mu_k) Z_k$ is called a reduced Hessian for the Lagrangian function estimated at $(\mathbf{x_k}, \mu_k)$.

The term of the reduced Hessian depends on the choice of the basis for the null space of the matrix $\nabla \mathbf{h}(\mathbf{x_k})^T$ and hence not unique. The reduced Hessian at $(\mathbf{x}^\star, \mu^\star)$ is positive definite and if $(\mathbf{x}_k, \mu_k)$ is chosen close to $(\mathbf{x}^\star, \mu^\star)$ the reduced Hessian at $(\mathbf{x}_k, \mu_k)$ will also be positive definite.

Let us decompose $\mathbf{d_x}$ as follow:

$$\mathbf{d_x} = Z_k p_z + Y_k p_y, \tag{2.67}$$

then the constraint in (2.48) becomes

$$\nabla \mathbf{h}(\mathbf{x_k})^{\mathbf{T}} Y_k p_y = -\mathbf{h}(\mathbf{x_k}), \tag{2.68}$$

and we can solve for p_y using

$$p_y = -[\nabla \mathbf{h}(\mathbf{x_k})^{\mathbf{T}} Y_k]^{-1} \mathbf{h}(\mathbf{x_k}). \tag{2.69}$$

The QP subproblem in (2.48) reduces to

$$\min_{\mathbf{p_z}} \quad \frac{1}{2} p_z^T Z_k^T B_k Z_k p_z + (\nabla f(\mathbf{x}_k^T + p_y^T B_k)) Z_k p_z \tag{2.70}$$

and p_z satisfies

$$p_z = -Z_k^T B_k Z_k[(\nabla f(\mathbf{x}_k) + p_y^T B_k)Z_k]. \tag{2.71}$$

The reduced matrix can be updated directly instead of updating B_k and recomputing $Z_k^T B_k Z_k$. Let R_k be a given $(n-m)\times(n-m)$ positive definite matrix at $\mathbf{x_k}$ and μ_k, we can compute the next solution by first computing p_y using (2.69) and setting

$$p_z = -R_k^{-1} Z_k^T (\nabla f(\mathbf{x_k}) + p_y^T B_k). \tag{2.72}$$

With p_z and p_y computed, the next iterate is computed by setting

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \mathbf{d_x}, \tag{2.73}$$

where $\mathbf{d_x}$ is given by (2.67). To continue the iteration, a new multiplier μ_{k+1} is needed and this can be computed using the least square formula (2.54). Also to update R_k, the BFGS scheme (2.59) can be employed by replacing B_k with R_k, and setting

$$\begin{aligned} S &= Z_k^T(\mathbf{x_{k+1}} - \mathbf{x_k}) = Z_k^T Z_k p_z \\ y &= Z_k^T[\nabla \mathcal{L}(\mathbf{x_k} + Z_k p_z, \mu_k) - \nabla \mathcal{L}(x_k, \mu_k)]. \end{aligned} \tag{2.74}$$

2.5.8 Merit function for SQP

Merit functions denoted Φ are often incorporated in SQP algorithms to monitor global convergence (see [8]). They are usually constructed in a way that the solutions of the nonlinear constrained problem are the unconstrained minimizers of the merit functions and satisfy

$$\Phi(\mathbf{x_k} + \alpha \mathbf{d_x}) < \Phi(\mathbf{x_k}), \tag{2.75}$$

where α is an appropriate steplength. Merit functions in constrained problems have parameters that control the step reduction of f and the feasibility of the solution $\mathbf{x_k}$ at each iteration.

To ensure a steady decrease of a merit function, the following conditions are imposed []

(i) The initial vector $\mathbf{x_0}$ and all subsequent iterates $\mathbf{x_k}$, $k \in \mathrm{N}$ are located in some compact set $\mathcal{C} \in \mathrm{R}^n$

(ii) The columns of $\nabla \mathbf{h}(\mathbf{x})$ are linearly independent for all $\mathbf{x} \in \mathcal{C}$.

We present the two commonly used merit functions below.

(i) **The augmented Lagrangian merit function**

The augmented merit function of the form [40, 8]

$$\Phi_F(\mathbf{x};\eta) = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^T \overline{\mu}(\mathbf{x}) + \frac{\eta}{2}\|\mathbf{h}(\mathbf{x})\|^2, \quad \eta > 0 \tag{2.76}$$

have been employed in literature. The multiplier $\overline{\mu}(\mathbf{x})$ can be estimated using the least square equation (2.54), and therefore $\mu(\mathbf{x}^\star) = \mu^\star$. Based on the above conditions, Φ_F and $\overline{\mu}$ are differentiable and Φ_F is bounded below on $\mathcal{C}$ for sufficiently large η. It has been proven that $\mathbf{x}^\star \in \mathcal{C}$ is a strong local minimum of Φ_F if and only if $\mathbf{x}^\star$ is a strong local minimum of the original nonlinear problem, provided that the above conditions are satisfied. However, if $\mathbf{x}$ is not a critical point of the original problem, then $\mathbf{d_x}$ is a descent direction for Φ_F.

The augmented merit function can be extended directly to handle inequality constraints in two ways. First by the active set strategy where some set of inequality constraints are treated as equality constraints. Let

$$I_k = \{i : g_i(\mathbf{x_k}) \geq \frac{-\lambda_i}{\eta}\}, \tag{2.77}$$

where at $\mathbf{x_k}$, I_k contains all unsatisfied constraints but no satisfied ones. The merit

function can then be defined as

$$\Phi_{FI}(\mathbf{x},\mu,\lambda;\eta) = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^T\mu(\mathbf{x}) + \frac{1}{2}\eta\|\mathbf{h}(\mathbf{x})\|^2 + \sum_{i\in I_k}(g_i(\mathbf{x})\lambda_i(\mathbf{x}) + \frac{1}{2}\eta g_i(\mathbf{x})^2) + \frac{1}{2\eta}\sum_{i\notin I_k}(\lambda(\mathbf{x}))_i^2. \tag{2.78}$$

The second extension uses the idea of slack variable formulation of the quadratic subproblem, where squared slack variables are added to $\mathbf{g}(\mathbf{x})$

$$\begin{aligned} \min_{\mathbf{x},\mathbf{t}} \quad & f(\mathbf{x_k}) \\ \mathbf{s.t} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) + \mathbf{t}^2 = \mathbf{0}. \end{aligned} \tag{2.79}$$

If we let $z_j = t_j^2$, for $j = 1,\ldots,m$, we have an adjusted equality constraint vector

$$\overline{h}(\mathbf{x},\mathbf{z}) = \begin{bmatrix} \mathbf{h}(\mathbf{x}) \\ \mathbf{g}(\mathbf{x}) + \mathbf{z}. \end{bmatrix} \tag{2.80}$$

The merit function is then constructed as

$$\Phi_{fz}(\mathbf{x},\mathbf{z}) = f(\mathbf{x}) + \overline{h}(\mathbf{x},\mathbf{z})^T\overline{\mu}(\mathbf{x},\mathbf{z}) + \frac{\eta}{2}\|\overline{\mathbf{h}(\mathbf{x},\mathbf{z})}\|^2, \tag{2.81}$$

where $\overline{\mu}(\mathbf{x},\mathbf{z})$ can be computed using the least square estimate (2.54) and the slack variables step $\mathbf{d_z}$ can be computed after each iteration using

$$\mathbf{d_z} = -(\nabla\mathbf{g}(\mathbf{x_k})^T\mathbf{d_x} + \mathbf{g}(\mathbf{x_k}) + \mathbf{z_k}). \tag{2.82}$$

Note that equation (2.82) is derived from the linearization of $\mathbf{g}(\mathbf{x}) + z = 0$.

(ii) **The ℓ_1 merit function** [22, 8] is another applicable merit function for the SQP algorithm. When the problem is defined with only equality constraints, we have it to

be

$$\Phi_1(\mathbf{x};\rho) = f(\mathbf{x}) + \rho\|\mathbf{h}(\mathbf{x})\|_1,\ \rho > 0 \tag{2.83}$$

while for the full constrained nonlinear problem we have it to be

$$\Phi_1(\mathbf{x}) = f(\mathbf{x}) + \rho[\|\mathbf{h}(\mathbf{x})\|_1 + \|\mathbf{g}^+(\mathbf{x})\|_1], \tag{2.84}$$

where

$$\mathbf{g}^+(\mathbf{x}) = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \leq 0 \\ g_i(\mathbf{x}) & \text{if } g_i(\mathbf{x}) > 0 \end{cases} \tag{2.85}$$

The ℓ_1 merit function is a form of penalty function just like the augmented Lagrangian. That is, there exists $\rho^\star$ such that for all $\rho \geq \rho^\star$, if $\mathbf{x}^\star$ is an unconstrained minimum of Φ_1 then $\mathbf{x}^\star$ is a solution of the nonlinear problem. Note that Eq. (2.83) is not differentiable on the feasible set and the term $\rho\|\mathbf{h}(\mathbf{x})\|_1$ cannot be squared because the exact penalty property will be lost. The merit function is constructed in such a way that as $\rho \to \infty$, the minimizer of Φ_1 converges to the solution of the nonlinear constrained problem.

Chapter 3

Minmax capacity provisioning on path networks

In this chapter, we develop and analyse a novel algorithm and data-structure suitable for evacuation procedure in remote and sparsely populated communities. When considering remotely or sparsely populated areas, such as the sparse areas of British Columbia, California and New South Wales, evacuation process may not be feasible by foot as assumed in many (traditional) sink location algorithms (see Chapter 1). As a matter of urgency, emergency vehicles or aircraft may have to be dispatched (or already situated) in order to evacuate the distressed population. In this case also, congestion may not be the primary concern of the evacuation planners and hence, there is need for a new evacuation procedure.

To explore this problem, we introduce the concept of capacity provisioning for evacuation on dynamic networks as an extension of the traditional sinks location problems, suitable for the effective evacuation planning of the aforementioned communities. Unlike the usual sink location problem (irrespective of the criterion) which always incorporate road capacities in the model development, this new concept involves the determination of the optimal capacities distribution from a fixed budgeted capacity under the assumption that the location of the sink is known. In other words, the budgeted capacity is viewed as a resource to be allocated optimally to the edges of the network such that the evacuation time of the evacuees to the sinks is minimized. We formerly present the problem definition and model in the next section, specifically for a path topology.

3.1 Model formulation and problem definition

Using the parameters defined in Chapter 1, We define our problem as follows:

Problem 3.1. Given a dynamic path network $\mathcal{N} = (P, c, \tau, \mathbf{w})$, a fixed sink $s \in P$, and a total capacity budget c. Let $z(\mathbf{x})$ denote the evacuation time which represents the time for the last evacuee to get to sink s given the capacity vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ where x_i represents the capacity assigned to edge e_i on the path. Find the optimal capacity vector $\mathbf{x}^\star$ such that $z(\mathbf{x}^\star)$ is minimum and $\sum_{i=1}^{n} x_i = c$.

Just like the traditional sink location problem on a path, we observe that two independent flows emerge on either side of the sink (see Chapter 1). If we denote by $z_L(\mathbf{x})$ (resp. $z_R(\mathbf{x})$) the evacuation time of evacuees on the left (resp. right) side of the sink, then $z(x) = \max\{z_L(\mathbf{x}), z_R\mathbf{x}\}$. We shall focus on function $z_L(\mathbf{x})$ and treat $z_R(\mathbf{x})$ symmetrically. For simplicity, we assume a path with n vertices with the sink s connected to vertex v_n via edge e_n. Thus, all evacuee vertices are on the left side of the sink. From Higashikawa et al. [24] and Bhattacharya et al. [5], we learn that the objective function or cost can be expressed in terms of n evacuation time functions $\theta_i(\mathbf{x})$

$$z_L(\mathbf{x}) = \max_{i \in [1,n]} \{\theta_i(x_i)\}, \tag{3.1}$$

where

$$\theta_i(x_i) = \frac{\sum_{j=1}^{i} w_j}{x(v_i, s)} + \tau d(v_i, s) \tag{3.2}$$

and $x(v_i, s)$ denotes the minimum edge capacity between a vertex v_i and the sink s, that is $x(v_i, s) = \min_{i \le j \le n} x_j$. The first term in Eq. (3.2) computes the waiting time (at v_i) of the last evacuee from v_1 if the congestion occurs at v_i. We next illustrate several simple instances of the capacity provisioning problem.

Illustration 3.1.0.1 (*Sink in the middle with two evacuee nodes*). Let us consider the simple path network shown in **Fig. 3.1**, where the sink is located at the middle of the path. We denote the budgeted capacity by c and the number of evacuees at v_1 and v_2 by w_1 and w_2

respectively. Our objective is to find optimal values of edge capacities x_1 and x_2 such that the completion time of all evacuees is minimized. Note also that $x_1 + x_2 = c$ and both edge capacities are non-negative. For simplicity, let $x_2 = x$ so that $x_1 = c - x$.

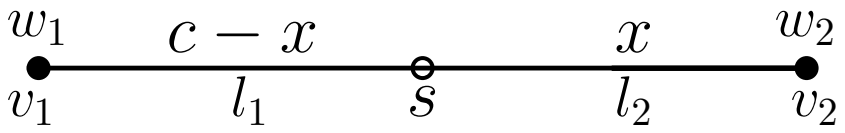

Figure 3.1: Simple path network with sink in the middle.

We understand that the evacuation times (or costs of evacuation) for evacuees on both sides of the sink are given by the following evacuation functions (see [24, 5]).

$$\begin{aligned} \theta_1(x) &= \frac{w_1}{c-x} + \tau l_1 \\ \theta_2(x) &= \frac{w_2}{x} + \tau l_2. \end{aligned} \tag{3.3}$$

Intuitively, the first term of the functions represents the time the last evacuee needs to wait at the node before it can proceed on the edge. This time is determined by the edge capacity. The second term represents the travel time through the edge. We notice that the optimal solution x^* has the following property: $\theta_1(x^*) = \theta_2(x^*)$. If this is not the case, assume $\theta_1(x^*) > \theta_2(x^*)$, thus $z(x^*) = \theta_1(x^*)$. Then there exists $\varepsilon > 0$ such that the solution $x^* - \varepsilon$, which is obtained by transferring ε capacity from edge (s, v_2) to edge (s, v_1), is still determined by $\theta_1(x^* - \varepsilon)$. But then $\theta_2(x^* - \varepsilon) \leq \theta_1(x^* - \varepsilon) = z(x^* - \varepsilon) < \theta_1(x^*) = z(x^*)$ which contradicts the assumption that x^* is optimal. Based on this property, we have the following quadratic equation

$$x^2(l_2 - l_1)\tau + x(w_1 + w_2 + \tau l_1 c - \tau l_2 c) - w_2 c = 0. \tag{3.4}$$

If $l_1 = l_2$, then we obtain $x_2 = x = \frac{w_2 c}{w_1 + w_2}$ and $x_1 = c - x = \frac{w_1 c}{w_1 + w_2}$.

Without loss of generality, suppose that $l_1 > l_2$. Then, the optimal solution is given by one of the roots of Eq. (3.4).

We can show that only one of the roots of the equation is feasible for our problem. The roots of Eq. (3.4) are

$$\begin{aligned} x_{a,b} &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} \pm \frac{1}{2}\sqrt{\left(\frac{w_1+w_2}{\tau(l_1-l_2)}+c\right)^2 - \frac{4w_2c}{\tau(l_1-l_2)}} \\ &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} \pm \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} + \frac{2(w_1-w_2)c}{\tau(l_1-l_2)}}. \end{aligned}$$

We analyse the first root $x = x_a$ as follow :

Case 1: When $w_1 > w_2$, we have

$$\begin{aligned} x_a &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} + \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} + \frac{2(w_1-w_2)c}{\tau(l_1-l_2)}} \\ x_a &> \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} + \frac{1}{2}\sqrt{\left(c + \frac{(w_1-w_2)}{\tau(l_1-l_2)}\right)^2} \\ &= c + \frac{w_1}{\tau(l_1-l_2)}. \end{aligned} \tag{3.5}$$

Case 2: When $w_1 < w_2$, we have

$$\begin{aligned} x_a &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} + \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} - \frac{2(w_2-w_1)c}{\tau(l_1-l_2)}} \\ x_a &> \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} + \frac{1}{2}\sqrt{\left(c - \frac{(w_2-w_1)}{\tau(l_1-l_2)}\right)^2} \\ &= c + \frac{w_1}{\tau(l_1-l_2)}. \end{aligned} \tag{3.6}$$

The case when $w_1 = w_2$ is obvious. Thus x_a is more than the total allocated budget c and is infeasible. The same idea can be employed to show that $0 < x_b < c$. We present the proof

for only the case $w_1 > w_2$, as the case $w_2 > w_1$ follows the same idea. We have

$$\begin{aligned} x_b &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} - \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} + \frac{2(w_1-w_2)c}{\tau(l_1-l_2)}} \\ &> \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} - \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} + \frac{2(w_1+w_2)c}{\tau(l_1-l_2)}} \\ &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} - \frac{1}{2}\sqrt{\left(c + \frac{(w_1+w_2)}{\tau(l_1-l_2)}\right)^2} = 0. \end{aligned} \tag{3.7}$$

Next we show that $x_b < c$

$$\begin{aligned} x_b &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} - \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} + \frac{2(w_1-w_2)c}{\tau(l_1-l_2)}} \\ &< \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} - \frac{1}{2}\sqrt{c^2 + \frac{(w_1+w_2)^2}{\tau^2(l_1-l_2)^2} - \frac{2(w_1+w_2)c}{\tau(l_1-l_2)}} \\ &= \frac{w_1+w_2}{2\tau(l_1-l_2)} + \frac{c}{2} - \frac{1}{2}\sqrt{\left(\frac{w_1+w_2}{\tau(l_1-l_2)} - c\right)^2} = c. \end{aligned} \tag{3.8}$$

Algorithm 2: Capacity allocation algorithm for dynamic path network with two evacuee nodes and a sink at the middle.

Result: Output optimal cost and vector of edge capacities for the network in Fig. 3.1

Inputs: w_1, w_2, c, τ, l_1, l_2;

if $l_1 = l_2$ **then**

  Assign $x_2^\star = \frac{w_2 c}{w_1+w_2}$;

else

  Assign $x_2^\star = x_b$, where x_b is the root of Eq. (3.4) that lies in $(0, c)$;

end

Compute $z(x_2^\star) = \theta_2(x_2^\star)$;

Output the optimal cost $z(x_2^\star)$ and vector $(c - x_2^\star, x_2^\star)$

Illustration 3.1.0.2 (*Path with two edges and sink at an endpoint*). Let us now consider

another simple path network (shown in **Fig. 3.2**)

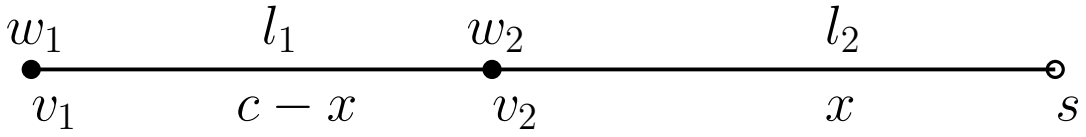

Figure 3.2: Illustration of capacities allocation on a two-edged path network with the sink located at the end vertex on the right.

where the sink is denoted by s on the right. The case where the sink is located on the left can be treated symmetrically. We again denote x_2 by x and $x_1 = c - x$. From the figure, the time taken for the first evacuee at v_2 to reach the sink is τl_2. Since the last evacuee at v_2 must wait up to $\frac{w_2}{x}$, then the evacuation time for the last evacuees at v_2 is $\frac{w_2}{x} + \tau l_2$. At the same time, evacuees from v_1 start moving towards the sink. We have to consider two cases:
Case 1: (Non-congestion scenario) Evacuees from v_1 arrive at v_2 and v_2 is empty (its evacuees have already left v_2). For this case, we can write the completion time as

$$z(x) = \theta_1(x) = \frac{w_1}{\min\{c-x, x\}} + \tau(l_1 + l_2). \tag{3.9}$$

Case 2: (Congestion scenario) Evacuees from v_1 get to v_2 but are being delayed due to backlog of evacuees waiting to evacuate through edge e_2.The completion time for this scenario is (see [5, 24])

$$z(x) = \theta_2(x) = \frac{w_1 + w_2}{x} + \tau l_2. \tag{3.10}$$

According to authors of [5] and [24], the two cases can be merged into a single relation:

$$z(x) = \max\{\theta_1(x), \theta_2(x)\}. \tag{3.11}$$

We observe the following property (see Theorem 3.2) of the optimal solution vector called the monotonicity property. The property states that the allocation of capacities must be non-decreasing towards the sink. This helps us further simplify the expression for the evacuation time. In addition, we now know that $x^\star \geq \frac{c}{2}$. We shall discuss the proof of the

property in Section 3.2.

Theorem 3.2. *Given a path P with $n+1$ vertices where $v_1,\dots v_n$ are occupied by evacuees $w_1,\dots,w_n$ and the sink is located at v_{n+1}. Let $\mathbf{x}^\star = (x_1^\star,\dots,x_n^\star)$ be the optimal edge-capacities vector, then $x_i^\star \geq x_j^\star$ if $i > j$.*

Based on Theorem 3.2, we rewrite both time functions as follows:

$$\begin{aligned} \theta_1(x) &= \frac{w_1}{c-x} + \tau(l_1+l_2) \\ \theta_2(x) &= \frac{w_1+w_2}{x} + \tau l_2. \end{aligned} \tag{3.12}$$

To obtain the optimal solution for $z(x)$ in Eq. (3.11) with θ_1 and θ_2 defined in Eq. (3.12), we first examine the nature of both evacuation time functions θ_1 and θ_2. We observe that evacuation function $\theta_1(x)$ (resp. $\theta_2(x)$) is increasing (resp. decreasing) with x and $\lim_{x\to c}\theta_1(x) = \infty$. Two situations are possible. The two evacuation functions may either intersect ($\theta_1(\frac{c}{2}) \leq \theta_2(\frac{c}{2})$) or not ($\theta_1(\frac{c}{2}) > \theta_2(\frac{c}{2})$) for $x \in [\frac{c}{2}, c]$. We explain the consequences of the two situations below:

Case 1 : If θ_1 and θ_2 do not intersect for $x \in [\frac{c}{2}, c]$, then, $z(x) = \theta_1(x)$ and $x = \frac{c}{2}$ is the optimal solution with the optimal cost $\theta_1(\frac{c}{2})$.

Case 2 : If θ_1 and θ_2 intersect, then the intersection point determines the optimal solution.

These two cases are determined by the values of θ_1 and θ_2 at point $\frac{c}{2}$. To identify these cases, we solve the following equation.

$$\theta_1\left(\frac{c}{2}\right) - \theta_2\left(\frac{c}{2}\right) = \tau l_1 - \frac{2w_2}{c} = 0, \tag{3.13}$$

and consequently we have

$$c = \frac{2w_2}{\tau l_1}. \tag{3.14}$$

We call the value of the capacity in Eq. (3.14) *critical capacity* and we denote it by $c_{critical}$.

The above shows that if we have any budgeted capacity c greater than $c_{critical}$, allocating

equal capacities is enough to attain optimality. However, if the two functions intersect at a point in $(c/2, c)$, then this point is the value of x that minimizes the cost function $z(x)$. To compute this point, we require $\theta_1(x) - \theta_2(x) = 0$, which leads to the following quadratic equation

$$\tau l_1 x^2 - (2w_1 + w_2 + \tau l_1 c)x + (w_1 + w_2)c = 0. \tag{3.15}$$

The solutions to Eq. (3.15) are

$$x_{a,b} = \frac{2w_1 + w_2 + \tau l_1 c}{2\tau l_1} \pm \sqrt{\frac{(w_2 - \tau l_1 c)^2 + 4w_1(w_1 + w_2)}{4\tau^2 l_1^2}}. \tag{3.16}$$

We now show in the lemma below that only one of the solutions in Eq. (3.16) lies in the open interval $(c/2, c)$ and the other can be discarded.

Lemma 3.3. *Given a path network with 3 vertices $\{v_1, v_2, v_3\}$ where the sink is located on v_3. If the total capacity c is less than the critical value $c_{critical}$, then the optimal solution is one of the zeros of Eq.* (3.15) *presented in Eq.* (3.16).

Proof. Based on the condition $\tau l_1 < \frac{2w_2}{c}$ from Eq. (3.14), we first show that the solution given by

$$x_a = \frac{2w_1 + w_2 + \tau l_1 c}{2\tau l_1} + \sqrt{\frac{(w_2 - \tau l_1 c)^2 + 4w_1(w_1 + w_2)}{4\tau^2 l_1^2}}. \tag{3.17}$$

is more than the budgeted capacity c to be shared. We analyse x_a as follow:

$$\begin{aligned}
x_a &= \frac{w_1}{\tau l_1} + \frac{w_2}{2\tau l_1} + \frac{c}{2} + \sqrt{\frac{1}{4}\left[\frac{w_2}{\tau l_1} - \frac{c}{1}\right]^2 + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{w_1 w_2}{(\tau l_1)^2}} \\
&\geq \frac{w_1}{\tau l_1} + \frac{c}{4} + \frac{c}{2} + \sqrt{\frac{1}{4}\left[\frac{c}{2} - c\right]^2 + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{c w_1}{2\tau l_1}} \\
&= \frac{3c}{4} + \frac{w_1}{\tau l_1} + \sqrt{\frac{c^2}{16} + \frac{w_1}{\tau l_1}\left[\frac{w_1}{\tau l_1} + \frac{c}{2}\right]} \\
&= \frac{3c}{4} + \frac{w_1}{\tau l_1} + \sqrt{\left(\frac{c}{4} + \frac{w_1}{\tau l_1}\right)^2} \\
&= c + \frac{2w_1}{\tau l_1},
\end{aligned} \tag{3.18}$$

Thus x_a is more than our budgeted capacity and not feasible.

With $c < \frac{2w_2}{\tau l_1}$, we analyse x_b as follow:

$$\begin{aligned} x_b &= \frac{w_1}{\tau l_1} + \frac{w_2}{2\tau l_1} + \frac{c}{2} - \sqrt{\frac{1}{4}\left[\frac{w_2}{\tau l_1} - \frac{c}{1}\right]^2 + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{w_1 w_2}{(\tau l_1)^2}} \\ &> \frac{w_1}{\tau l_1} + \frac{w_2}{2\tau l_1} + \frac{c}{2} - \sqrt{\frac{1}{4}\left[\frac{w_2}{\tau l_1} - \frac{2w_2}{\tau l_1}\right]^2 + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{w_1 w_2}{(\tau l_1)^2}} \\ &= \frac{w_1}{\tau l_1} + \frac{w_2}{2\tau l_1} + \frac{c}{2} - \sqrt{\left(\frac{w_2}{2\tau l_1} + \frac{w_1}{\tau l_1}\right)^2} = \frac{c}{2}. \end{aligned} \tag{3.19}$$

Next,(with $c < \frac{2w_2}{\tau l_1}$) we show that $x_b < c$ as follow:

$$\begin{aligned} x_b &= \frac{w_1}{\tau l_1} + \frac{w_2}{2\tau l_1} + \frac{c}{2} - \sqrt{\frac{1}{4}\left[\frac{w_2}{\tau l_1} - \frac{c}{1}\right]^2 + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{w_1 w_2}{(\tau l_1)^2}} \\ &< \frac{w_1}{\tau l_1} + \frac{c}{4} + \frac{c}{2} - \sqrt{\frac{1}{4}\left[\frac{c}{2} - \frac{c}{1}\right]^2 + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{w_1 c}{2\tau l_1}} \\ &= \frac{w_1}{\tau l_1} + \frac{3c}{4} - \sqrt{\frac{c^2}{16} + \left(\frac{w_1}{\tau l_1}\right)^2 + \frac{w_1 c}{2\tau l_1}} \\ &< \frac{w_1}{\tau l_1} + \frac{3c}{4} - \sqrt{\frac{c^2}{16} + \left(\frac{w_1}{\tau l_1}\right)^2 - \frac{w_1 c}{2\tau l_1}} \\ &= \frac{w_1}{\tau l_1} + \frac{3c}{4} - \sqrt{\left(\frac{w_1}{\tau l_1} - \frac{c}{4}\right)^2} = c. \end{aligned} \tag{3.20}$$

Since $x_b \in (\frac{c}{2}, c)$, $x = x_b$ is the optimal solution and the optimal solution vector is $\mathbf{x}^\star = (c - x_b, x_b)$. □

Definition 3.4. Whenever the optimal capacities for two consecutive edges are not obtainable by allocating equal capacity, we call this situation a touching scenario, and the solution method a touching solution. The solution x_b in Eq. (3.19) is an example of a touching solution.

We can summarize our findings in the following algorithm for the case of a path network with two evacuee nodes and the sink node located at one of the endpoints of the path.

Algorithm 3: Capacity allocation algorithm for dynamic path network with two evacuee nodes and one sink at an endpoint.

Result: Output optimal cost and vector of edge capacities for a two-edged network

Inputs: w_1, w_2, c, τ, l_1, l_2;

Compute the critical capacity $c_{critical} = \frac{2w_2}{\tau l_1}$;

if *If* $c \geq c_{critical}$ **then**

  Assign equal capacity $x^\star = c - x^\star = \frac{c}{2}$;

else

  Let $x^\star = x_b$, where x_b is computed in (3.19);

end

Compute $z(x^\star) = \theta_1(x^\star)$;

Output the cost z and vector $(c - x^\star, x^\star)$

Illustration 3.1.0.3 (*Path network with four evacuee nodes, two nodes on each side of the sink*). Let us consider a more interesting case where we have a path network with four edges with the sink located in the middle of the path (see **Fig. 3.3**).

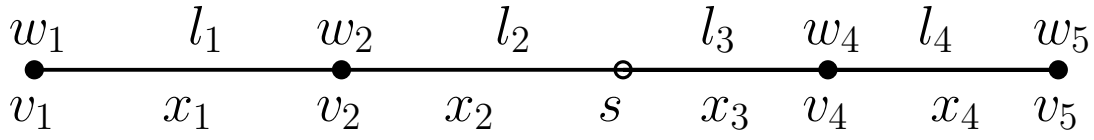

Figure 3.3: Four-edged path network with sink at the middle.

Based on Theorem 3.2, we require a non-decreasing allocation of capacities towards the sink from both sides. In other words, we require $x_2 \geq x_1$ and $x_3 \geq x_4$. From the path graph, we have the following evacuation functions

$$\begin{aligned} z_L(\mathbf{x^T}) &= z_{1,2}(\mathbf{x}^T) = \max\{\theta_1(x_1), \theta_2(x_2))\} \\ z_R(\mathbf{x^T}) &= z_{3,4}(\mathbf{x}^T) = \max\{\theta_3(x_3), \theta_4(x_4))\}, \end{aligned} \tag{3.21}$$

where

$$\begin{aligned}
\theta_1(x_1) &= \frac{w_1}{x_1} + \tau(l_1 + l_2) \\
\theta_2(x_2) &= \frac{w_1 + w_2}{x_2} + \tau l_2 \\
\theta_3(x_3) &= \frac{w_3 + w_4}{x_3} + \tau l_3 \\
\theta_4(x_4) &= \frac{w_4}{x_4} + \tau(l_3 + l_4).
\end{aligned} \tag{3.22}$$

Before we present an algorithm for this problem, we prove the following lemma which gives us an insight into how the optimal solutions can be obtained.

Lemma 3.5. *At the optimal solution* $\mathbf{x}^\star$*,*

$$z_{1,2}(\mathbf{x}^\star) = z_{3,4}(\mathbf{x}^\star). \tag{3.23}$$

Proof. Suppose the above is not true and without loss of generality, let us assume $z_{1,2}(\mathbf{x}^\star) > z_{3,4}(\mathbf{x}^\star)$. Then there exists $\varepsilon > 0$ so that we decrease x_3 and x_4 by ε each and increase x_1 and x_2 by ε each to obtain a new vector $\mathbf{x}$ satisfying the conditions $z_{1,2}(\mathbf{x}) \geq z_{3,4}(\mathbf{x})$ and $z_{1,2}(\mathbf{x}) < z_{1,2}(\mathbf{x}^*)$. This contradicts the fact that $\mathbf{x}^\star$ is the optimal solution. □

Based on the lemma above, we sketch an algorithm to solve this more general problem. We observe that, each sub-path on either side of the sink reduces to a problem which we know how to solve if we are given the total budget capacity for the sub-path. Let X_1 be the total capacity allocated to one of the sub-paths, which we call the left sub-path. We extend our notation to denote the optimal solution cost for the left sub-path if the total capacity allocated to the left sub-path is X_1 by $z_L(X_1)$. Similarly, we have $z_R(c - X_1)$ as the optimal cost for the right sub-path. In addition, the actual solutions $x_1, \ldots, x_4$ are parametric solutions with X_1 as the parameter. Algorithm 4 illustrates the steps we can take to solve this problem instance.

Algorithm 4: Capacity allocation algorithm for the instance in **Fig. 3.3**

Result: Output optimal cost $z_L(X_1^*)$ and vector $\mathbf{x}^*$

Inputs: w_1, w_2, w_3, w_4 c, τ, l_1, l_2, l_3, l_4;

Compute the critical capacities $c_{1,2} = \frac{2w_2}{\tau l_1}$ and $c_{3,4} = \frac{2w_4}{\tau l_4}$;

Solve the left sub-path problem with budget $X_1 = c_{1,2}$ and the right with budget $X_2 = c - X_1$;

if *If* $z_L(X_1) \geq z_R(X_2)$ **then**

$X_1^\star \geq c_{1,2}$ and $x_1^\star = x_2^\star = \frac{X_1}{2}$ is the parametric solution;

$z_L(X_1) \leftarrow \frac{2w_1}{X_1} + \tau(l_1 + l_2)$;

else

$X_1^\star < c_{1,2}$ and $z_L(X_1) \leftarrow f_2(x_b)$, where $x_2^* = x_b$ (given by (3.19)) is a parametric solution that depends on X_1 and $x_1^* = X_1 - x_2^*$;

end

Solve the left sub-path problem with cost $X_1 = c - c_{3,4}$ and the right with cost $X_2 = c_{3,4}$;

Repeat the test for the right sub-path;

Obtain function $z_R(X_1)$ and the corresponding parametric solutions x_3^* and x_4^* as functions of X_1;

Solve equation $z_L(X_1) = z_R(X_1)$ numerically, and obtain X_1^*;

Output the optimal cost $z_L(X_1^\star)$ and the capacity vector $(x_1^\star, x_2^\star, x_3^\star, x_4^\star)$ by evaluating the parametric solutions at X_1^*;

Before we present the general properties of the optimal solution for a given arbitrary path network, let us briefly discuss the running time of Algorithm 4. We note that, in the first part of the algorithm, we obtain a parametric optimal solution $\mathbf{x}^*$ which depends on the unknown capacity budget X_1 assigned to the sub-path left of the sink. The time complexity for this computation is $O(1)$. The more challenging phase of the algorithm is to determine the value for the budget X_1 so that $z_L(X_1) = z_R(X_1)$. The expressions $z_L(X_1)$ and

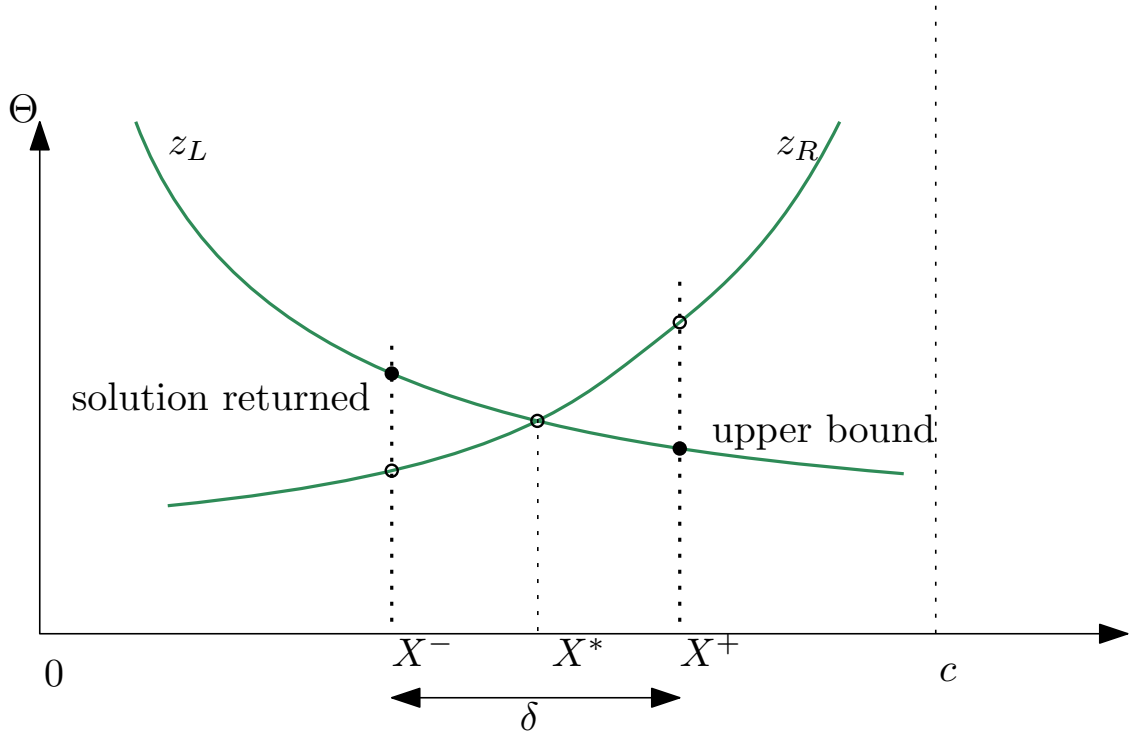

Figure 3.4: Binary search procedure for the optimal capacity budget X_1.

$z_R(X_1)$ are linear functions of the reciprocal of the parametric solution. In the worst case, the parametric solutions depend on budget X_1 as illustrated by Eq. 3.16. It is impractical to attempt to find an algebraic expression for X_1 that solves equation $z_L(X_1) = z_R(X_1)$. This equation can be transformed into a complicated polynomial equation of degree larger than four in X_1 and we know that there is no algebraic expression for the roots of a general polynomial equation of degree higher than four [1][1]. We opt to find the value for X_1 approximately using a simple binary search procedure. We find next an upper bound on the number of steps needed by this binary search procedure to obtain a solution with evacuation time $Z \leq (1+\varepsilon)Z^*$ for any $\varepsilon > 0$, where Z^* is the evacuation time of the optimal solution. Let δ be the difference between the upper and lower bounds on the optimal X_1^* that is maintained by the binary search procedure. Since the initial binary search range for X_1 is interval $(0,c)$ where c is the value of the total capacity budget, the number of binary search iterations performed is $O(\log \frac{c}{\delta})$. Let X_1^- and X_1^+ be the lower and upper bounds respectively

[1]This doesn't mean that an algebraic solution for this particular equation does not exist.

on the optimal X_1^* maintained by the binary search (see Fig. 3.4). We know that $z_L(X_1)$ is a decreasing function of X_1 and $z_R(X_1)$ is increasing. During any iteration of the binary search procedure, value $\min\{z_L(X_1^-), z_R(X_1^+)\}$ can be returned as an approximate solution and $\max\{z_R(X_1^-), z_L(X_1^+)\}$ is a lower bound on the optimal solution. The binary search can stop when $\min\{z_L(X_1^-), z_R(X_1^+)\} \leq (1+\varepsilon)\max\{z_R(X_1^-), z_L(X_1^+)\}$. It can be verified that this condition is satisfied if,

$$z_R(X_1^- + \delta) \leq (1+\varepsilon) \cdot z_R(X_1^-) \quad \text{or} \tag{3.24}$$

$$z_L(X_1^-) \leq (1+\varepsilon) \cdot z_L(X_1^- + \delta). \tag{3.25}$$

We consider first function $z_L(X_1)$ and condition (3.25) to identify a suitable constraint on δ. To simplify the calculations, we note from Algorithm 3 that there are two cases for calculating the capacity assignment of the left sub-path. In both cases, the capacity assigned to an edge such as (v_2, s) in Fig. 3.2 for the left sub-path grows linearly with the budget X_1 (we note that the unknown parameter X_1 corresponds to budget c in Eq. (3.16)). The evacuation time $z_L(X_1)$ grows asymptotically as a fractional expression in X_1. To simplify the calculations, we take $z_L(X_1) = \frac{a}{gX_1+h}$ for some constants a, g, h with $a, g > 0$. If we substitute $z_L(X_1^-)$ in (3.25), we obtain

$$\frac{a}{gX_1^- + h} \leq (1+\varepsilon)\frac{a}{g(X_1^- + \delta) + h}, \ \ldots \tag{3.26}$$

$$\delta \leq \varepsilon \frac{gX_1^- + h}{g} = \varepsilon X_1^- + \varepsilon \frac{h}{g}. \tag{3.27}$$

Similarly, we can write $z_R(X_1) = \frac{a'}{h' - g'X_1}$ which we substitute in (3.24).

$$\frac{a'}{h' - g'X_1^+} \leq (1+\varepsilon)\frac{a'}{h' - g'(X_1^+ - \delta)}, \ \ldots$$

$$\delta \leq \varepsilon \frac{h' - g'X_1^+}{g'} = \varepsilon \frac{h' - g'(X_1^- - \delta)}{g'} = \varepsilon \frac{h' - g'X_1^-}{g'} + \varepsilon\delta.$$

We consider the tighter inequality that implies the inequality above,

$$\delta \leq \varepsilon \frac{h' - g' X_1^-}{g'} = \varepsilon \frac{h'}{g'} - \varepsilon X_1^-. \tag{3.28}$$

We notice that the upper bounds on δ from Eq.(3.27) and (3.28) can be arbitrarily small depending on the value of X_1^-. However, the sum of the upper bounds, $\varepsilon(\frac{h}{g} + \frac{h'}{g'})$, does not depend on X_1^-. This means that, when $\delta \leq \frac{\varepsilon}{2}(\frac{h'}{g'} + \frac{h}{g})$, at least one of eq. (3.24) and (3.25) is satisfied and we obtain a $(1+\varepsilon)$ approximate solution. We conclude that, the running time of Algorithm 4 is $O(\log \frac{c}{\varepsilon})$.

3.2 Properties of the optimal solution for an arbitrary path with the sink located at an endpoint

In this section, we study and analyse the properties of the optimal solution vector $\mathbf{x}^\star$ exploitable to develop an efficient algorithm given an arbitrary path with the sink located at the endpoint. Note that $\mathbf{x}^\star$ is a vector of capacities which, when allocated to the edges of the path network, minimizes the completion time and also satisfies the capacity budget condition $\mathbf{x}^\star \cdot \mathbf{1} = c$, where $\mathbf{1}$ is the vector with all elements equal to 1. We recall the definition of evacuation functions $\theta_i(x_i)$ for the case when the sink is located on point v_{n+1}, the right endpoint of the network

$$\theta_i(x_i) = \frac{W[1,i]}{x(v_i,s)} + \tau L[i], \tag{3.29}$$

where $W[1,i] = \sum_{j=1}^{i} w_j$ and $L[i] = \sum_{j=i}^{n} l_j$. We also let $L[1,i] = \sum_{j=1}^{i} l_j$.

The first property of the optimal solution is the monotonicity property given in Theorem 3.2. We have exploited and employed this property in Illustration 3.1.0.2 and 3.1.0.3. We present the proof below.

Proof of Theorem 3.2. Let us assume that the optimal solution, $\mathbf{x}^\star$ is not monotonic in-

creasing towards the sink at v_{n+1}. Then there exist $x_i > x_{i+1}$ in the ordering of $\mathbf{x}^\star$. Let $\mathbf{x}'$ be another vector with the same elements as $\mathbf{x}^\star$, except that $x'_{i+1} = x_i$ and $x'_i = x_{i+1}$. We notice that $\theta_j(\mathbf{x}') = \theta_j(\mathbf{x}^\star)$ for all $j \leq i$ and $j \geq i+2$. We observe that

$$\theta_{i+1}(\mathbf{x}^\star) = \frac{W[1,i+1]}{\min\{x_{i+1}, x(v_{i+2},s)\}} + \tau L[i+1], \tag{3.30}$$

and

$$\begin{aligned} \theta_{i+1}(\mathbf{x}') &= \frac{W[1,i+1]}{\min\{x'_{i+1}, x(v_{i+2},s)\}} + \tau L[i+1] \\ &= \frac{W[1,i+1]}{\min\{x_i, x(v_{i+2},s)\}} + \tau L[i+1]. \end{aligned} \tag{3.31}$$

Since $x_i > x_{i+1}$, then $\min\{x_i, x(v_{i+2},s)\} \geq \min\{x_{i+1}, x(v_{i+2},s)\}$ and $\theta_{i+1}(\mathbf{x}') \leq \theta_{i+1}(\mathbf{x}^\star)$. Therefore, it is either the cost is the same and $\mathbf{x}'$ is OPT or cost decreases which is a contradiction that $\mathbf{x}^\star$ is OPT. □

Based on Theorem 3.2, we have the following definition which redefines the evacuation time for any index i in terms of the discovered minimum capacity and therefore definition in Eq.(3.2) changes.

Definition 3.6. Given an instance of a capacity provisioning problem on a path network with a sink located at the right end, the associated evacuation time for an index i is given by

$$\theta_i(x_i) = \frac{W[1,i]}{x_i} + \tau L[i]. \tag{3.32}$$

The second theorem gives us an adjustment to the monotonicity property and shows the possibility of assigning equal capacity to adjacent edges. The theorem also gives us a candidate of the optimal cost amongst many of the time functions.

Theorem 3.7. *Let P be a path network with $n+1$ vertices, with the sink located at the $(n+1)$th vertex. Let $h^\star$ be the largest index defining the optimal cost $z^\star$, i.e.,*

$$z^\star = \theta_{h^\star} = \frac{W[1,h^\star]}{x_{h^\star}} + \tau L[h^\star].$$

Then the optimal capacity vector has the following properties

(i) $x^\star_{h^\star} = x^\star_{h^\star+1} = \ldots = x^\star_n$

(ii) *Let* $i_1, i_2, i_3 \ldots, i_k$ *be the indices that define* $z^\star$*, enumerated in increasing order, i.e.* $i_1 < i_2 < i_3 \ldots < i_k$ *and* $\theta_{i_1} = \theta_{i_2} = \theta_{i_3} \ldots = \theta_{i_k} = \theta_{h^\star}$*. Then,* $i_1 = 1$*. Moreover, if* $i_{j+1} > i_j + 1$ *for some* $1 \leq j < k$*, then* $x_{i_j} = x_{i_j+1} = \ldots = x_{i_{j+1}-1}$ *for all* $j \in \{1, \ldots k-1\}$.

This theorem states two important properties of the optimal edge capacity vector. First, the evacuation time is always determined by the evacuation function θ_1. Second, if other evacuation functions are equal in value with θ_1, namely $\theta_{i_2}, \ldots, \theta_{i_k}$, then these indices determine the only increasing sequence of elements in the optimal capacity vector.

Proof. Suppose there exists an index $j \geq h^\star$ such that $x_{j+1} > x_j$. By definition, $\theta_j > \theta_{j+1}$. Therefore, we can write

$$\begin{aligned} \frac{W[1,j]}{x_j} + \tau L[j] &> \frac{W[1,j+1]}{x_{j+1}} + \tau L[j+1] \\ \frac{W[1,j]}{x_j} + \tau l_j &> \frac{W[1,j+1]}{x_{j+1}}. \end{aligned} \tag{3.33}$$

This means that we can decrease the capacity x_{j+1} by an appropriate value $\varepsilon > 0$ in such a way that inequality $\theta_{j+1} < \theta_j$ is maintained. We then distribute the extra capacity ε to all the other edges on the path, namely to $x_1, \ldots, x_j$ and to $x_{j+2}, \ldots x_n$, in such a way as to maintain the monotonicity property of the vector of capacities. As a consequence, all of the evacuation functions $\theta_1, \ldots \theta_j$ and $\theta_{j+2} \ldots \theta_n$ decrease. Therefore $z^\star$ decreases which is a contradiction.

To prove the second part of the theorem, we use the diagram in **Fig. 3.5**.

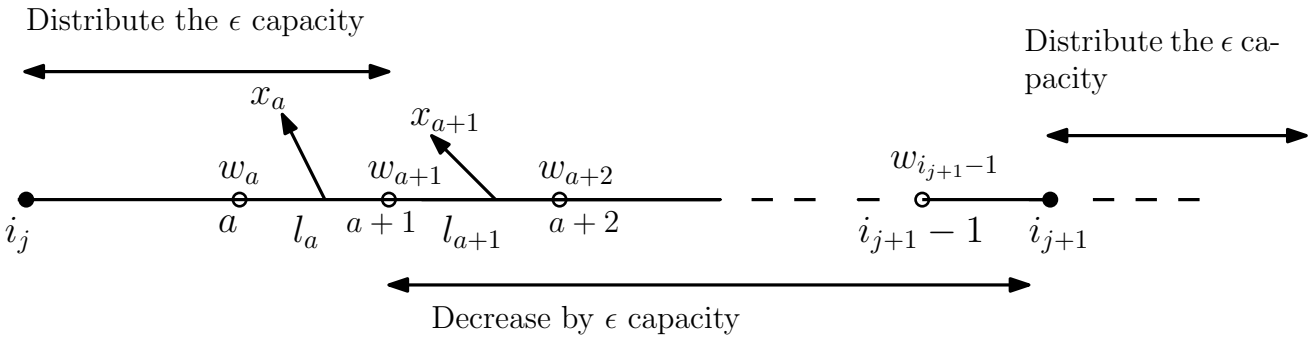

Figure 3.5: Illustration for the proof of part (ii) from Theorem 3.7.

Suppose $x_a < x_{a+1}$ for some $i_j \leq a < i_{j+1} - 1$. Then we can take an appropriate amount $\varepsilon > 0$ of capacity from $x_{a+1}, x_{a+2}, \ldots, x_{i_{j+1}-1}$ and distribute it uniformly to all of the other edges, namely $x_1, \ldots x_a$ and $x_{i+1}, \ldots x_n$ without violating the monotonicity property. As a consequence, θ_p for all $p \in \{a+1, \ldots, i_{j+1}-1\}$ increase while the other values of θ_p for all $p \notin \{a+1, \ldots, i_{j+1}-1\}$ decrease. If ε is appropriately chosen, $z^\star$ decreases, a contradiction. Moreover, if $\theta_1 < \theta_{i_1}$, then we can take an $\varepsilon > 0$ capacity from x_1 and add it uniformly to all other capacities so that $\theta_1 \leq \theta_p$ and θ_p decreases for all $p \geq 2$. Consequently $z^\star$ decreases, which is also a contradiction. Therefore $i_1 = 1$. □

3.3 Capacity provisioning for an arbitrary path network with the sink fixed at an endpoint

We first consider the path illustrated in **Fig. 3.6**, where we have three vertices with evacuees and the sink is located at the right endpoint.

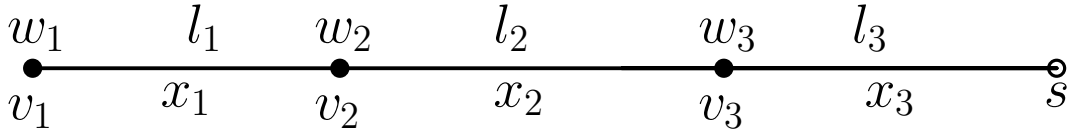

Figure 3.6: Example to illustrate longer path network with sink at the end.

Just as before, we need to minimize the following objective function

$$z(\mathbf{x}) = \max\{\theta_1, \theta_2, \theta_3\}, \tag{3.34}$$

where

$$\begin{aligned}
\theta_1 &= \frac{w_1}{x_1} + \tau L[1] \\
\theta_2 &= \frac{W[1,2]}{x_2} + \tau L[2] \\
\theta_3 &= \frac{W[1,3]}{x_3} + \tau l_3.
\end{aligned}$$

Based on Theorem 3.2, we require $x_3 \geq x_2 \geq x_1$.

We can formulate our problem as a convex, non-linear program, with variables Z, x_1, x_2, x_3 and use a general purpose solver to find the optimal solution:

$$\begin{aligned}
&\min Z, \text{ subject to:} \\
&Z \geq \frac{w_1}{x_1} + \tau L[1] \\
&Z \geq \frac{W[1,2]}{x_2} + \tau L[2] \\
&Z \geq \frac{W[1,3]}{x_3} + \tau l_3. \\
&x_1 + x_2 + x_3 = c \\
&0 \leq x_1 \leq x_2 \leq x_3.
\end{aligned}$$

However, we can avoid solving this non-linear program by exploiting Theorem 3.7. We propose a combinatorial algorithm for an arbitrary path with the sink fixed at an endpoint, that will output an optimal parametric capacity assignment to the edges of the path. More precisely, we obtain the solutions $x_2^*(x_1), \ldots, x_n^*(x_1)$ as function of the parameter x_1, the capacity assigned to the first edge. We then determine the optimal value for x_1 by solving

numerically the budget capacity equation,

$$x_1 + x_2(x_1) + \ldots + x_n(x_1) = c. \tag{3.35}$$

Moreover, we also obtain specific conditions for Eq. (3.35), which allow us to determine x_1 to the desired precision, by binary search.

According to Theorem 3.7, θ_1 must be a candidate that determines the optimal cost. If θ_2 is also a candidate then we can write, after some basic algebraic manipulations,

$$\theta_1 = \theta_2 \Longrightarrow \frac{1}{x_2} = \frac{w_1}{w_1 + w_2}\frac{1}{x_1} + \frac{\tau l_1}{w_1 + w_2}. \tag{3.36}$$

We can substitute $\frac{1}{x_i} = y_i$ so that Eq. (3.36) becomes

$$y_2 = \frac{w_1}{w_1 + w_2} y_1 + \frac{\tau l_1}{w_1 + w_2}. \tag{3.37}$$

Since Theorem 3.2 requires $x_1 \leq x_2$, this implies $y_2 \leq y_1$. Thus, we have

$$\frac{w_1}{w_1 + w_2} y_1 + \frac{\tau l_1}{w_1 + w_2} \leq y_1 \Longrightarrow y_1 \geq \frac{\tau l_1}{w_2}. \tag{3.38}$$

The above implies $x_1 \leq \frac{w_2}{\tau l_1}$ and as long as this is true, we can always use Eq. (3.36) to solve for x_2. Therefore we have following:

$$y_2 = \begin{cases} y_1 & \text{if } y_1 < \frac{\tau l_1}{w_2} \\ \frac{w_1}{w_1+w_2} y_1 + \frac{\tau l_1}{w_1+w_2} & \text{if } y_1 \geq \frac{\tau l_1}{w_2} \end{cases} \tag{3.39}$$

We illustrate the parametric solution y_2 in **Fig. 3.7**

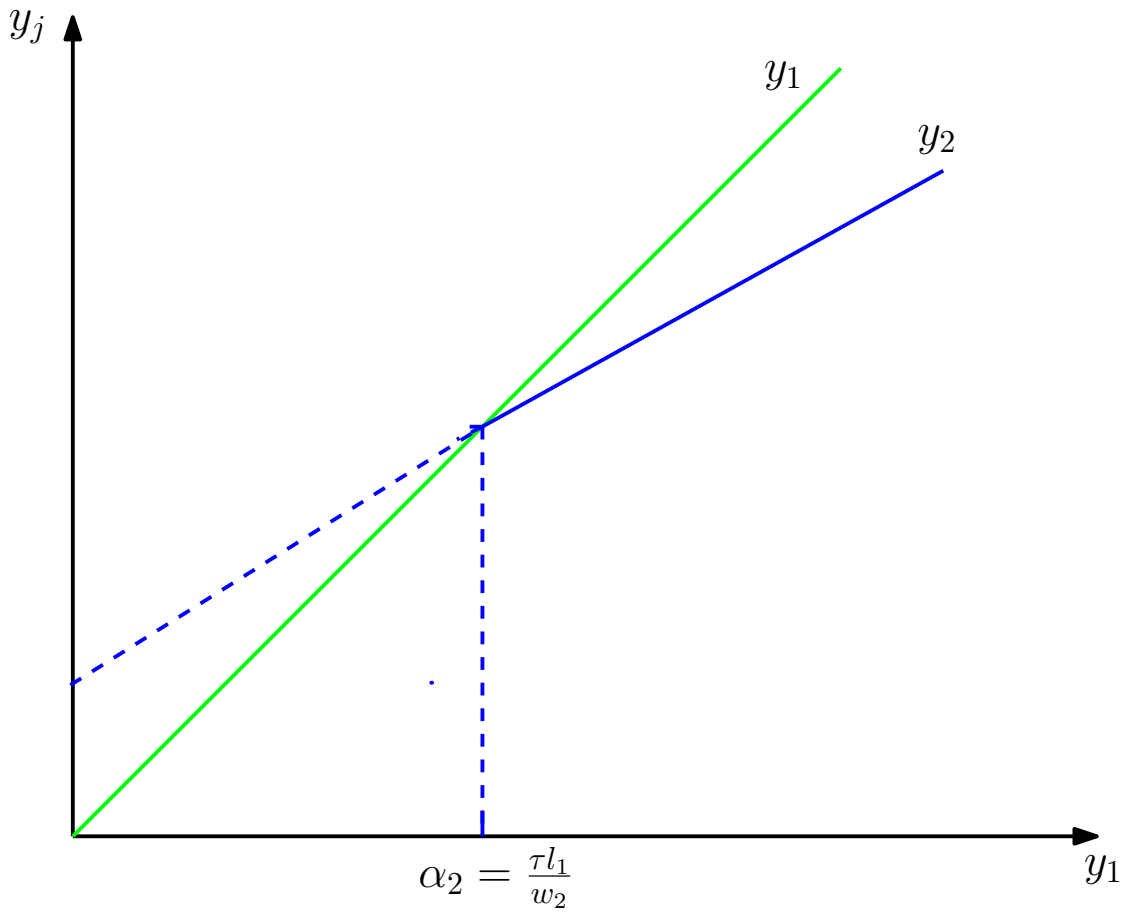

Figure 3.7: Graph illustrating the optimal computation of x_2.

For example, let us recall the two nodes network discussed in Illustration 3.1.0.2. If y_1 is in the region below α_2, then we just need to solve for y_1 in $\frac{2}{y_1} = c$. Otherwise, we need to solve for y_1 in $\frac{w_1+w_2}{y_1(2w_1+w_2)+\tau l_1} = \frac{1}{c}$, since $\frac{1}{x_1+x_2} = \frac{1}{c}$.

We notice that y_2 is a concave piece-wise linear function of y_1. In general, we can compute y_i for all $i \geq 2$ as a function of y_1 using Theorems 3.7 and 3.2 as follows. We consider two possible cases:

Case 1: If θ_i is dominated, that is $\theta_i < \theta_1$, then $x_i = x_{i-1}$, according to Theorem 3.7.

Case 2: If $\theta_i = \theta_1$, then θ_i is a candidate of the optimal cost. From the relation $\theta_i = \theta_1$ and with x_i substituted by $\frac{1}{y_i}$, we obtain

$$y_i' = \frac{w_1}{W[1,i]} y_1 + \tau \frac{L[1,i-1]}{W[1,i]}. \tag{3.40}$$

Putting the two cases together, we have the following:

Lemma 3.8. *Capacity for each edge e_i for $i \geq 2$ can be computed using*

$$y_i = \begin{cases} \frac{w_1}{W[1,i]} y_1 + \tau \frac{L[1,i-1]}{W[1,i]}, & \text{if } y_1 \geq \alpha_i \\ y_{i-1}, & \text{otherwise,} \end{cases} \tag{3.41}$$

where α_i is the value of y_1 for which $y_i = y_{i-1}$.

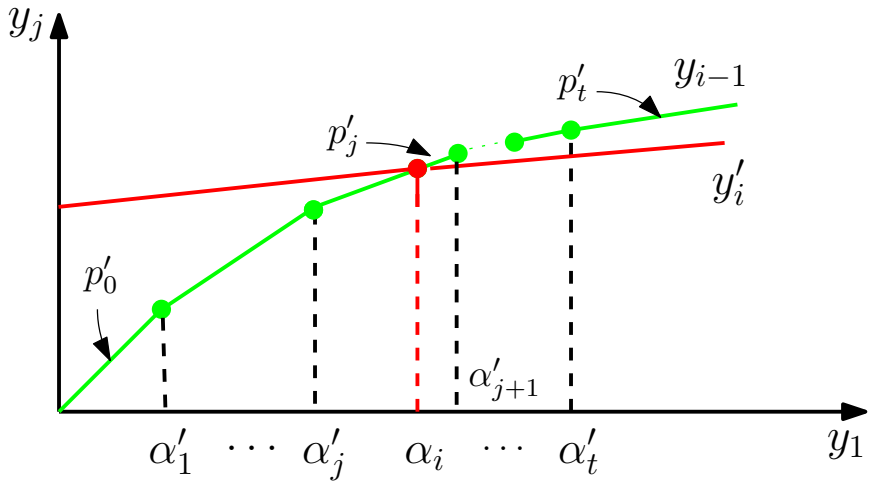

Figure 3.8: Explanation of how α_i is computed. We let $\alpha'_1, \ldots, \alpha'_t$ be the breakpoints of y_{i-1} and $p'_0, \ldots, p'_t$ the corresponding linear pieces. Only the line segment $[\alpha_i, \infty)$ is stored in the segment tree.

To construct an efficient data structure for the solution of our problem, we exploit the property of the linear function derived in Case 2 above (see Eq. (3.40)). Let $\alpha'_1, \ldots, \alpha'_t$ be the breakpoints of y_{i-1} and let $p'_0, \ldots, p'_t$ be the corresponding linear pieces (see **Fig. 3.8**). We denote by α_i the value of y_1 for which $y'_i = y_{i-1}$ (the intersection point between y'_i and y_{i-1}). First, we argue that this intersection point is unique, because the slope of the linear function y'_i, $\frac{w_1}{W[1,i]}$ is smaller than the slope of the last linear piece of y_{i-1}, $\frac{w_1}{W[1,i-1]}$. This property is crucial for the computation of all intersection points as well as the development of our data structure as we shall see later in Subsection 3.3.1

Once we have the relation (3.41) for all y_i with $2 \leq i \leq n$, all we need is to numerically solve the following equation in y_1:

$$\frac{1}{y_1} + \frac{1}{y_2} + \ldots + \frac{1}{y_n} = c. \tag{3.42}$$

We now only have to discuss the procedure to compute α_i and how we store the y_i functions so that Eq. (3.42) can be solved efficiently.

3.3.1 Implementation details and data structure

Suppose the piece-wise linear function y_{i-1} is known and its linear pieces are available in order. We will compute y_i from y_{i-1} using Eq. (3.41). To find the intersection point α_i, we first test the intersection with linear piece p'_t (see **Fig. 3.8**). There are two cases. (a) Function y'_i intersects linear piece p'_t at a point smaller than breakpoint α'_t, and (b) y'_i and p'_t intersect at a point greater than or equal to breakpoint α'_t. Since the slope of function y'_i is smaller that that of p'_t, case (a) means that $y'_i < p'_t$ (y'_i dominates y_{i-1}) and thus p'_t is not part of function y_i. We repeat the test with linear piece p'_{t-1}, and so on, until we find the linear piece p'_j whose intersection with y'_i falls in case (b). That last intersection point is breakpoint α_i we needed to compute. We remark that computing breakpoint α_i takes amortized $O(1)$ time since any linear piece whose intersection falls in case (a) is tested at most once for all $i \in \{1, \ldots, n\}$.

The procedure illustrated above will compute the parametric solution y_i for all $i \in \{2, \ldots, n\}$ in total time $O(n)$, however, we must be careful how we store this parametric solution. The trivial approach of storing y_i functions separately has $O(n^2)$ time and space complexity. We propose to store the parametric solutions in a segment tree [28] (see Illustration 3.3.1.1). The data structure has $O(n)$ space complexity and can be easily constructed in time $O(n \log n)$, as described next.

1) Compute all of the breakpoints α_i, $i = 2, \ldots n$ associated with functions y_i. We note that these values may not be obtained in order. This step takes $O(n)$ time as previously

explained.

2) To construct the segment tree, we sort the elements α_i, say in increasing order. For each element α_i sorted, we make sure to retain the corresponding value of the index i. This step takes $O(n \log n)$ time.

3) We construct the segment tree over the α_i sorted elements, in $O(n)$ time.

4) For each $i \in \{2, \ldots, n\}$, we insert interval $[\alpha_i, \infty)$ in the segment tree, in total $O(n \log n)$ time (see **Fig. 3.8** for an example).

We can now explain how we can use this segment tree to evaluate the sum $\frac{1}{y_1} + \frac{1}{y_2} + \ldots + \frac{1}{y_n}$ at some fixed value $y_1 = \alpha'$, in $O(\log n + n) = O(n)$ time (see Eq. (3.42)). For simplicity, consider the example in **Fig. 3.9** and **Fig. 3.10**.

In this example, the query value α' is between breakpoints α_4 and α_3 and the vertical query line through α' intersects the intervals in the segment tree corresponding to indices 2, 4, and 7. This means that y_2 and y_3 are evaluated using linear function y'_2; y_4, y_5, y_6 are evaluated using linear function y'_4; and y_7 is evaluated using linear function y'_7.

In general, we describe the following procedure to evaluate the sum of reciprocals of the parametric solutions y_i:

a) Given query value α', identify two consecutive breakpoints $\alpha_i < \alpha' <= \alpha_j$ in the segment tree. Total time, $O(\log n)$.

b) Traverse the segment tree from the leaf node of α_j to the root and collect the indices of all intersected intervals. Total time: $O(\log n + n) = O(n)$. Let $i_1 < i_2 < \ldots < i_k$ be these indices in order.

c) We note that the list of indices of intersected intervals may not be obtained in the desired order from the segment tree. We can sort these indices using counting sort in total time $O(n)$ [14].

d) If there are two or more indices: for any two consecutive indices i_j and i_{j+1}, evaluate $y_{i_j}, \ldots, y_{i_{j+1}-1}$ using linear function y'_{i_j}. If there are one or more indices: evaluate $y_1, \ldots y_{i_1-1}$ using y_1. If i_1 does not exist: then evaluate all y_i parametric solutions using y_1.

Illustration 3.3.1.1. We show below (see **Fig. 3.9**) an example for arbitrary seven y'_i functions together with their breakpoints.

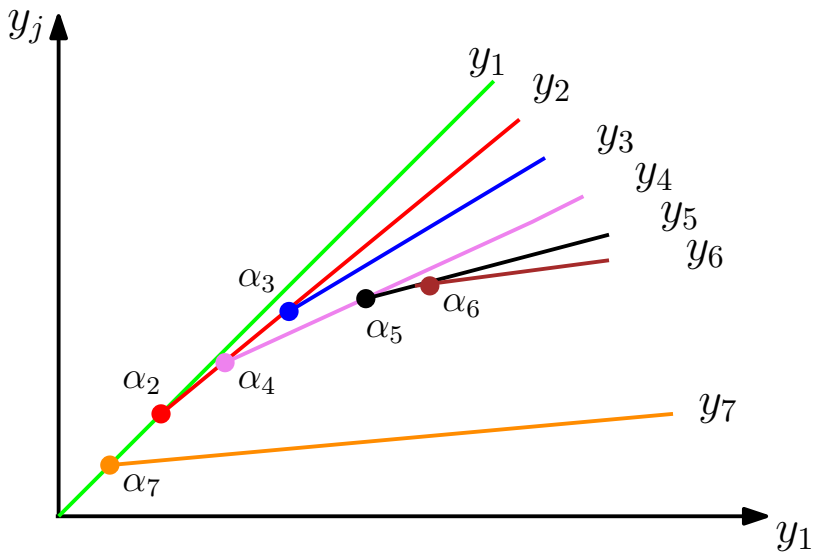

Figure 3.9: Seven arbitrary y'_i functions with their corresponding breakpoints. Only one-sided line segments $[\alpha_i, \infty]$ are stored in the segment tree.

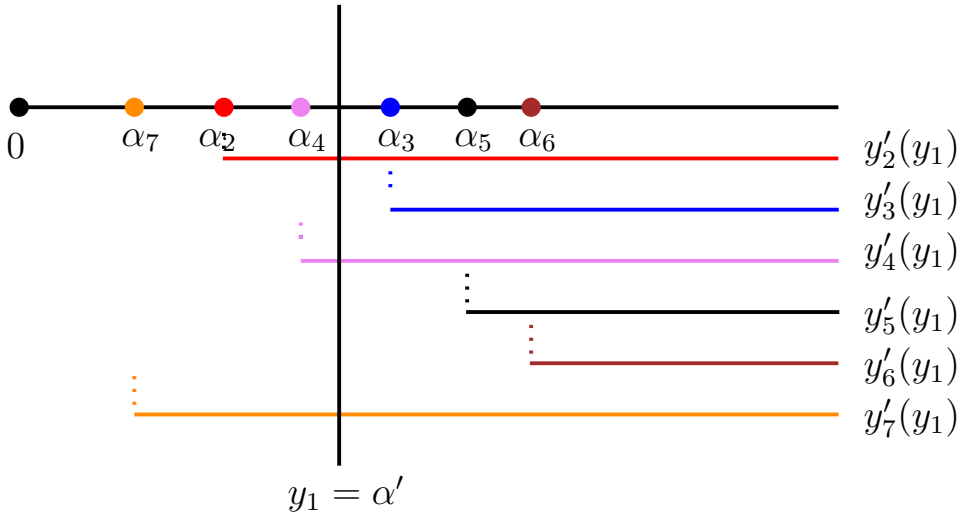

Figure 3.10: Stored line segments $[\alpha_i, \infty)$ for y_i functions in Fig. 3.9. The vertical line shows an arbitrary query point $y_1 = \alpha'$ that cuts through only 3 line segments.

Fig. 3.10 shows the line segments stored in the segment tree together with a vertical line that shows how an arbitrary query point $y_1 = \alpha'$ goes through just 3 line segments. From **Fig. 3.10**, we notice that segment y_2' and y_3' share the same segment of y_2' and segments y_4', y_5' and y_6' share the same line segment of y_4' based on the query point $y_1 = \alpha'$. For this example, we resolve Eq. (3.42) in the binary search as follow:

$$\frac{1}{\alpha'} + \frac{2}{y_2'(\alpha')} + \frac{3}{y_4'(\alpha')} + \frac{1}{y_7'(\alpha')} \leq (\geq) c \tag{3.43}$$

Using our segment tree, we can now proceed with binary search to identify a single interval for y_1 that contains the optimal value y_1^* in such a way that all y_i functions are linear in this interval. Thus, Eq. (3.42) consists of a sum of $O(n)$ reciprocals of linear functions. With the segment tree, this binary search procedure takes $O(n \log n)$ time.

We present the algorithm in Algorithm 5. In the algorithm, we let $\mathbf{W}$ be the sorted vector of weights on the vertices and $\mathbf{L}$ denotes the vector of edge lengths.

Algorithm 5: Capacity allocation algorithm for an arbitrary path network with a sink at its end

Result: Output the optimal cost and vector of edge capacities for an arbitrary path network with the sink at its end

Inputs: $\mathbf{W}$, $\mathbf{L}$, c, τ;

Compute all breakpoints α_i of y_i', for $i = 2, \ldots n$ and store them in a stack ;

Sort the $n-1$ breakpoints and construct a segment tree on them ;

Store only the needed $n-1$ intervals $[\alpha_i, \infty)$ at the nodes of the tree in the usual way;

Execute Binary Search to find the interval I^* for y_1^* and obtain the reciprocal functions for y_i ;

Solve Eq. 3.42 numerically by binary search over interval I^* and obtain y_1^*;

Output the solutions $\frac{1}{y_i(y_1^*)}$;

Output the cost $z = \theta_1(x_1^\star)$

We now summarize the time complexity of algorithm 5 in the following theorem:

Theorem 3.9. *The optimal capacities vector* $\mathbf{x}^\star$ *for a path network of* $n+1$ *vertices with a sink at its end can be obtained in* $O(n\log n + n\log(c/\xi))$ *time, where* ξ *is the precision for parameter* y_1^*.

Proof. The first term in the complexity of the algorithm follows from the earlier discussion. For the second term, we argue that each binary search iteration is resolved by evaluating $\frac{1}{y'} + \frac{1}{y_2(y')} + \cdots + \frac{1}{y_n(y')} \leq (\geq) c$ for some value y' in interval I^*. Alternatively, a numerical scheme (e.g., Newton method) or binary search procedure can be employed to determine $y_1^\star$ to some chosen precision ξ. Note that the range for I^* is of order of c, hence we can run the search operation at a cost of $O(\log(c/\xi))$ rounds. Since it takes $O(n)$ to evaluate all the linear functions y_i, it follows that the whole operation takes $O(n\log(c/\xi))$. □

3.4 Capacity provisioning for evacuation of an arbitrary path with one arbitrary sink fixed

We can extend Algorithm 5 to solve the general case. First, we compute, independently of each other, the segment trees for each sub-path on either side of the sink. Let A_L and A_R be the list of α values for the left and right sub-paths respectively. Let $\theta_1^L(x_1)$ and $\theta_1^R(u_1)$ be the corresponding candidates of the costs for both subnetworks according to Theorem 3.7, so that $z_L(X^\star) = \theta_1^L(x_1^\star)$ and $z_R(U^\star) = \theta_1^R(u_1^\star)$, where $X^\star$ (resp. $U^\star$) is the sum of optimal capacities for the left (resp. right) subpath. If we choose a query value $x_1 = \frac{1}{y_1}$ for the left subpath, we can easily solve for u_1 using $\theta_1^L(\frac{1}{y_1}) = \theta_1^R(\frac{1}{u_1})$. That is, for a fixed sink s, we have

$$\frac{1}{u_1} = \frac{w_1^L y_1}{w_1^R} + \frac{\tau}{w_1^R}(L^L[1] - L^R[1]), \tag{3.44}$$

where w_1^L (resp. w_1^R) is the weight on the first vertex v_1^L (resp. v_1^R) on the left (resp. right) subpath, $L^L[1] = d(v_1^L, s)$ and $L^R[1] = d(v_1^R, s)$. Now, we can execute Algorithm 5 on the left subpath with query $\frac{1}{y_1}$, compute the total capacity U of the right subpath using Eq. (3.41) and u_1 given by Eq. (3.44). The addition of assigned capacities for both left and right is then compared with c and the binary search procedure continues based on the comparison. This procedure takes $O(n \log n + n \log(c/\xi))$ time.

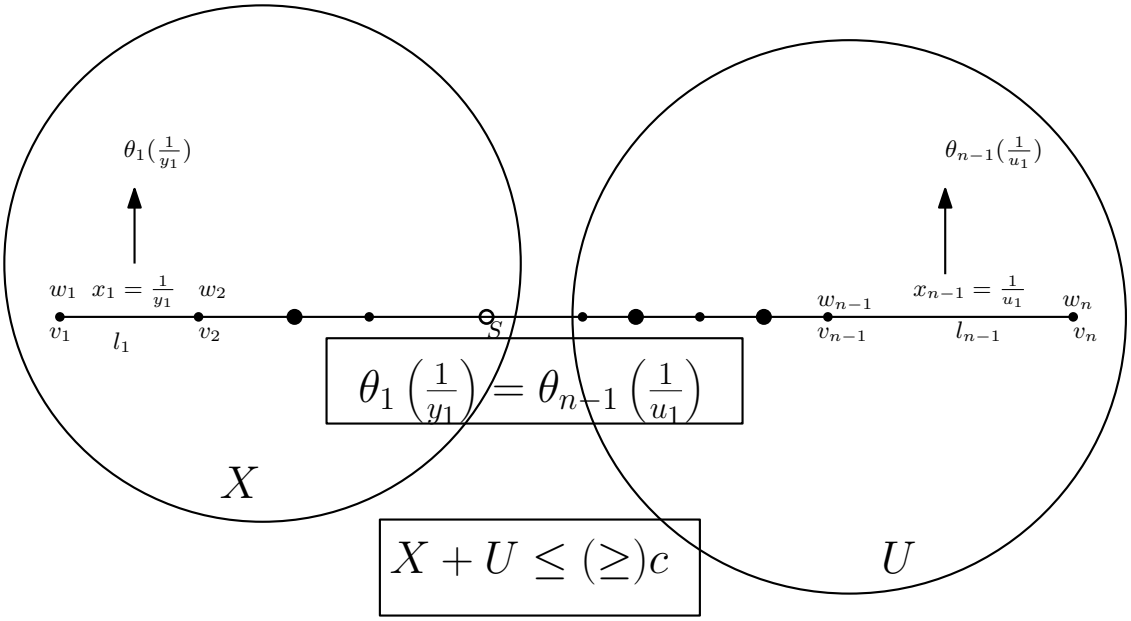

Figure 3.11: Illustration of the application of Algorithm 5 to an arbitrary path network with a fixed sink.

Chapter 4

Capacity provisioning on star networks

The vital idea we learnt from allocating capacity on path networks is that, if we can find the optimal allocation for the furthest edge from the sink, we can find the optimal allocation for all other edges in the network using the parametric equation given in (3.40). In this chapter, we shall look at how we can extend this idea to star networks where the sink is located in the middle of the star. We shall also investigate a complicated case where the sink is located on a leaf node. For the latter case, we shall exploit the flow theorem idea of Klinz employed in [10] to tackle the complicated case. We restrict our studies for the latter case to a star network with three links (or edges), as more properties are required to tackle the general case. The next section presents the case where the sink is located in the centre of the star graph.

4.1 Star network with a sink in the middle

We investigate the problem of optimal allocation of capacities in star topology (see Figure 4.1), where the sink is located at the centre. This can be linked in reality to a group of scattered communities linked together by a star topology with individual edges linked to a safe-haven at the centre

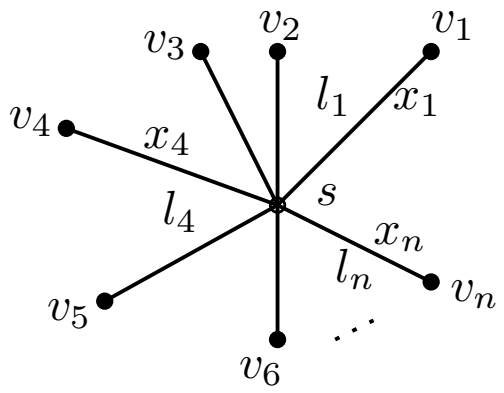

Figure 4.1: Simple star graph with sink in the middle.

We observe that the monotonicity property in 3.2 does not apply here as each vertex has a direct link to the sink. It is however easy to construct the model for the time functions associated with the edges of the network. That is, the cost functions for all $1 \leq i \leq n$ is given by

$$\theta_i(x_i) = \frac{w_i}{x_i} + \tau l_i. \tag{4.1}$$

Unlike in the path network studied earlier where the time function associated with the furthest edge always dominate other cost functions at optimality, this case is different as given below in Theorem 4.1

Theorem 4.1. *When the capacity allocation is optimal,* $\theta_1(x_1) = \theta_2(x_2) = \cdots = \theta_n(x_n)$.

We present the proof below.

Proof. Suppose there is an optimal allocation for which $\theta_j(x_j) < \theta_i$ for some indices i and j. Then we can take an appropriately small amount $\varepsilon > 0$ from the capacity x_j and distribute it equally to the other edges of the star graph, thus decreasing $\theta_i(x_i)$ and all θ functions other than θ_j. Therefore $\max_{1\leq i\leq n} \theta_i(x_i)$ decreases and so the evacuation time also decreases, which is a contradiction. □

4.1.1 Algorithm for star networks with a sink in the middle

Without loss of generality, we consider the leaf nodes of the star network in Figure 4.1 labelled in non-increasing order of their edge length, so $l_1 \geq l_2 \geq \ldots \geq l_n$. From the relation $\theta_i(x_i) = \theta_1(x_1)$, we have the following linear function:

$$y_i = \frac{w_1}{w_i} y_1 + \tau \frac{l_1 - l_i}{w_i}, \ \forall 1 \leq i \leq n, \tag{4.2}$$

where we denote $y_i = \frac{1}{x_i}$ for all $1 \leq i \leq n$. Notice that the second term in Eq.(4.2) is nonnegative and that is the reason for the edge length ordering. These equations describe an optimal parametric solution to the capacity allocation problem when the sink is located at the central vertex. We can obtain the optimal value for y_1 using Eq. (3.42) and the technique discussed in Subsection 3.3.1. We let $\mathbf{L}$ be the sorted edge length according to the labelling described above, and $\mathbf{W}$ be the corresponding weight vector. We present the algorithm below:

Algorithm 6: Capacity allocation algorithm for star networks (Figure 4.1) with a sink at its centre.

Result: Output the optimal cost and vector of edge capacities for the star network 4.1

Inputs: $\mathbf{W}$, $\mathbf{L}$, c, τ;

Employ the technique discussed in Subsection 3.3.1 ;

Output the solutions $\frac{1}{y_i(y_1^*)}$;

Output the cost $z = \theta_1(x_1^\star)$

4.2 Star networks with more edges and a sink located at the centre

We extend our study to a more complex star network which have more edges in its links but the sink is still fixed at the centre (see Figure 4.2)

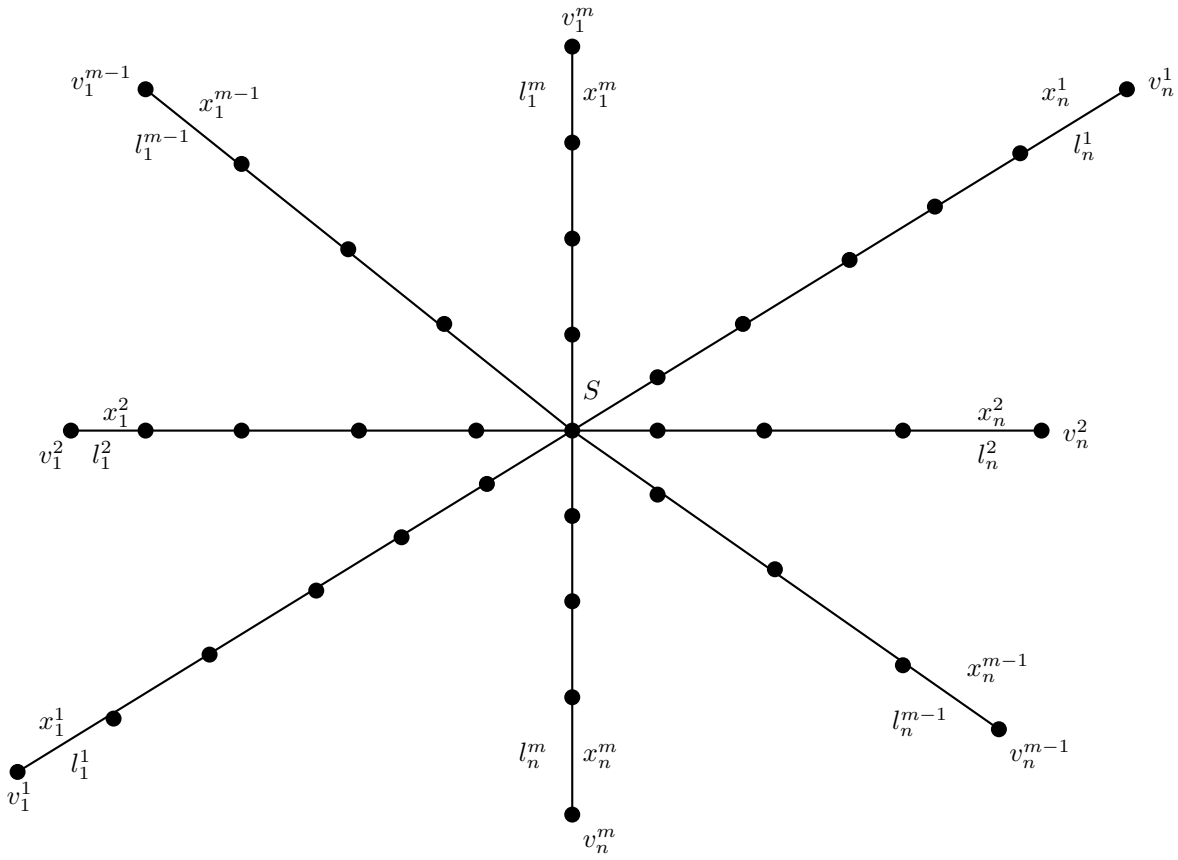

Figure 4.2: Regular star network with more edges and sink at the centre

To solve the above problem, we realise that the balancing algorithm discussed in Illustration 3.1.0.3 must come into play. Let us assume that v_1^1 is the furthest node to the sink amongst all the leaf nodes, then at optimal solution, we require $\theta_1^1 = \theta_1^j$ for $j = 2,\ldots,m$. We also understand that $\theta_1^1 = \theta_n^j$ for $j = 2,\ldots,m$ at optimal solution. Therefore, we can write

$$\theta_1^1 = \theta_1^2 = \cdots = \theta_1^m = \theta_n^1 = \theta_n^2 = \cdots = \theta_n^m. \tag{4.3}$$

More to the properties of the optimal solution, and precisely for the interior edges, we understand that the monotonicity property (see 3.2) must be observed toward the sink. The above equation helps us to employ the same idea of restricting the problem to finding the optimal allocation to the furthest edge provided we can compute all other edge capacities using this optimal value. Let $y_i^j = \frac{1}{x_i^j}$ and from Eq.(4.3), we can write $y_n^1, y_n^2, \ldots, y_n^m$ as

functions of $y_1^1, y_1^2, \ldots, y_1^m$. That is,

$$\begin{aligned} y_n^1 &= g_1(y_1^1) \\ y_n^2 &= g_2(y_1^2) \\ &\vdots \quad \cdots \quad \vdots \\ y_n^m &= g_m(y_1^m). \end{aligned} \tag{4.4}$$

With θ_1^1 being the function associated with the furthest edge to the sink, we can compute $x_1^j = \frac{1}{y_1^j}$ and $x_n^j = \frac{1}{y_n^j}$, $j = 2,3,\ldots,m$ as functions of y_1^1. That is

$$\begin{aligned} y_1^2 &= h_1(x_1^1) \\ y_1^3 &= h_2(x_1^1) \\ &\vdots \quad \cdots \quad \vdots \\ y_1^m &= h_{m-1}(y_1^1) \end{aligned} \tag{4.5}$$

and

$$\begin{aligned} y_n^2 &= q_1(x_1^1) \\ y_n^3 &= q_2(x_1^1) \\ &\vdots \quad \cdots \quad \vdots \\ y_n^m &= q_{m-1}(y_1^1). \end{aligned} \tag{4.6}$$

Note also that the interior edge capacities $y_2^1, \ldots, y_{n-1}^1$ can be written as functions of y_1^1, $y_2^2, \ldots, y_{n-1}^2$ as functions of y_1^2 and $y_2^m, \ldots, y_{n-1}^m$ as functions of y_1^m. That is,

$$\begin{aligned} y_2^1 = r_1^1(y_1^1) \quad & y_2^2 = r_1^2(y_1^1) \cdots y_2^m = r_1^m(y_1^m) \\ y_3^1 = r_2^1(y_1^1) \quad & y_3^2 = r_2^2(y_1^2) \cdots \quad y_3^m = r_2^m(y_1^m) \\ & \vdots \quad \cdots \quad \vdots \\ y_{n-1}^1 = r_{n-2}(y_1^1) \quad & y_{n-1}^2 = r_{n-2}^2(y_1^2) \cdots \quad y_{n-1}^m = r_{n-2}^m(y_1^m). \end{aligned} \tag{4.7}$$

Thus all unknown edge capacities can be written as functions of the furthest edge capacity y_1^1 and we can write

$$\begin{aligned} & y_1^1 + h_1(y_1^1) + \cdots + h_{m-1}(y_1^1) + q_1(y_1^1) + \cdots + q_{m-1}(y_1^1) \\ & + r_1^1(y_1^1) + \cdots + r_{n-1}^1(y_1^1) + r_1^2(h_1(y_1^1)) + \cdots + r_{n-2}^2(h_1(y_1^1)) \\ & + \cdots + r_{n-2}^2(h_1(y_1^1)) + \cdots + r_1^m(h_{n-1}(y_1^1)) + \cdots + r_{n-2}^m(h_{n-1}(x_1^1)) + \cdots + r_{n-2}^m(h_{m-1}(x_1^1)) = c, \end{aligned} \tag{4.8}$$

and the binary search of the previous chapter (see 3.3.1) can employed.

4.3 Three edged star network with the sink located on a leaf

We extend our study of capacity provisioning on star networks, where in this case the sink is located on a leaf node (see Figure 4.3)

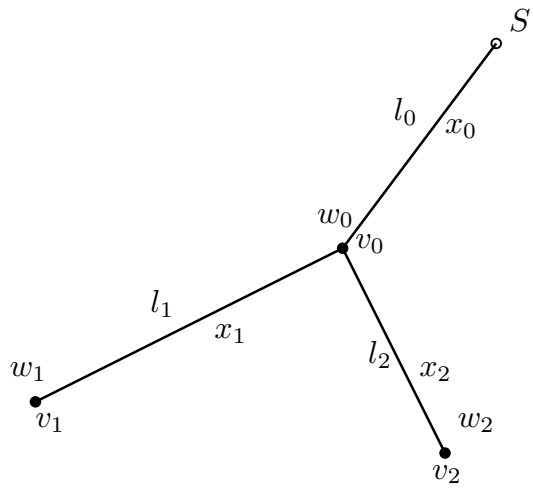

Figure 4.3: Three-edged tree network with a sink at its leaf

This case is more difficult compared to the cases of the star networks considered earlier as it was easy to compute all other edge capacities in term of the furthest edge capacity, and the binary search technique discussed in Subsection 3.3.1 can be employed to search for the optimal value of the furthest edge capacity that solves the allocation problem and such that the addition of the capacity is equal to the budgeted capacity c.

Before we proceed, we present the following theorem called the Klinz theorem em-

ployed in [10] for the quickest transshipment problem. This is because the evacuation problem is a special case of the quickest transshipment problem [10, 25]. The theorem gives us an idea of computing the optimal cost of flow:

Theorem 4.2 (Klinz theorem [10, 26])**.** *Let S be the set of terminals (set of supply sources and sink) and let A be a subset of S. Let $o(A)$ denote the maximum amount of flow that the sources (of supply) in A can send to the sink in $S \setminus A$ in time θ without considering other terminals and let $v(A)$ be the total supply in A. Then an evacuation problem is feasible if and only if $o(A) \geq v(A)$.*

Therefore given a dynamic tree network with evacuees on vertices, the optimal cost of flow is determined by the largest evacuation time when all the subsets that constitute the powerset of evacuees vertices are considered. From the tree network above, the powerset consists of 7 elements given by $\{v_0, v_1, v_2, \{v_1, v_2\}, \{v_1, v_0\}, \{v_2, v_0\}, \{v_1, v_2, v_0\}\}$. However, some of them can be discarded as they are dominated by obvious ones. We note that the flow from $\{v_1, v_0\}, \{v_2, v_0\}$ and v_0 are dominated by the flow from v_1, v_2 and $\{v_1, v_2, v_0\} = v_{012}$ respectively. Therefore, we need to consider the flow dictated by the following subsets $\{v_1, v_2, v_{12}, v_{012}\}$ and the one with the maximum evacuation time is taken as the optimal cost. In addition to the property of the optimal solution and cost, the following lemma emphasizes a monotone allocation from the edges associated with the leaves to the sink:

Lemma 4.3. *Capacity allocation is monotone non-decreasing from the leave edges to the sink.*

Proof. The flow from v_1 and v_2 to the sink s can be seen as two path networks and therefore the same argument of Theorem 3.2 of a path network applies. □

Let us denote by f_1 (resp. f_2) the amount of flow from v_1 (resp. v_2) that the network can allow to the sink within time t. According to Klinz theorem, we let θ_1 (resp. θ_2) to denote the evacuation time that matches up with the flow from v_1 (resp. v_2) and the corresponding

unknown edge capacity is denoted by x_1 (resp. x_2). Let us assume that v_1 is the furthest to the sink. The flow dictated by $\{v_1, v_2\}$ denoted by $f_{1,2}$ starts with capacity x_2 because the first evacuee arrives from v_2 and at $\tau(l_0 + l_1)$, we have evacuees arriving from v_1 and thus more flow. However, the rate of this flow is limited by the capacity x_0 of the link e_0 or $x_1 + x_2$. Using Klinz theorem, we look at the weights w_1 and w_2 and check the time for which the flow f_{12} equals $w_1 + w_2$. We denote this evacuation time by θ_{12}.

4.3.1 Algorithm strategy

We note by Lemma 4.3 that $\frac{c}{3} \leq x_0 < c$. We fix a value for x_0 in its range so that $x_1 + x_2 = c - x_0$. Then we look for the best way to determine x_1 and x_2 such that $\max\{\theta_1, \theta_2, \theta_{12}\}$ is minimized. We then use this approach as a subroutine to determine x_0 in such a way that the optimal solution or cost is attained.

Observation. If x_0 is large such that $x_0 > x_1 + x_2$ then $x_0 > \frac{c}{2}$. Then the best assignment is the one in which $\theta_0 = \theta_1 = \theta_2 = \theta_{12}$. Any solution that deviates from this equal cost will make one of the evacuation time higher and such evacuation time can be reduced by adjusting the capacities appropriately.

Note that it is possible to have $x_0 > \frac{c}{2}$ because of the possibility of a large weight at v_0 and it is reasonable to assign a large capacity to the link e_0 so that many evacuees can be evacuated and also reduce congestion at v_0.

Case 1: Based on the observation if x_0 is fixed in $[\frac{c}{2}, c)$, then $x_1 + x_2$ is constant. Let θ be the time for which all evacuation time $\theta_2 = \theta_1 = \theta_{12}$. Suppose we have θ_1 and θ_2 such that θ does not exist, we can consider the addition of x_1 and x_2 and construct the solution that gives θ.

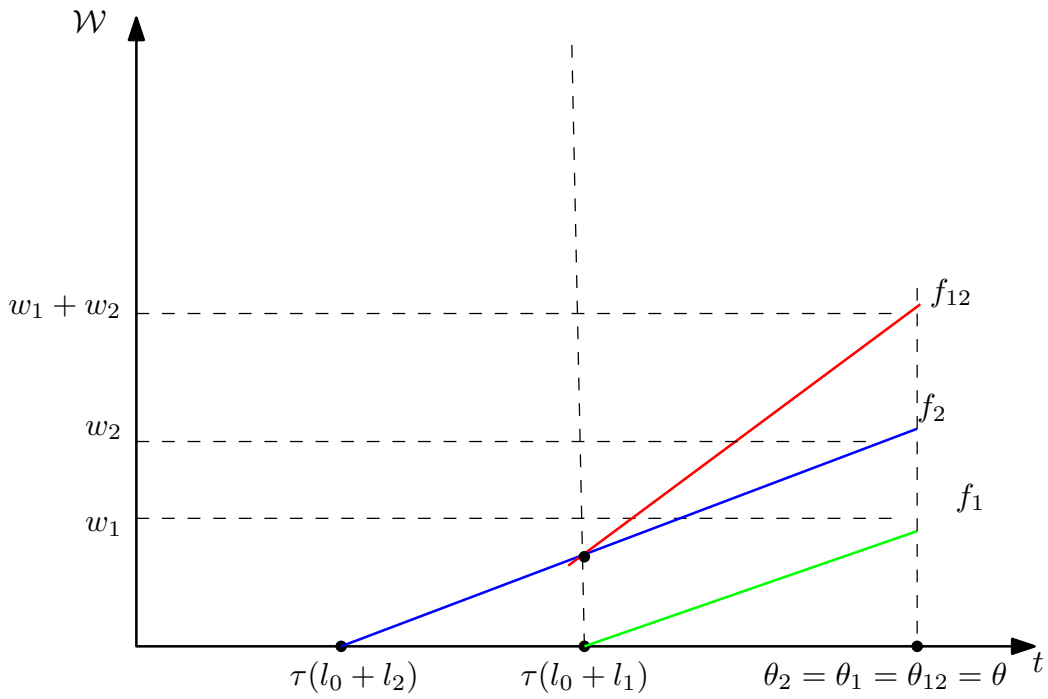

Figure 4.4: Graph illustrating the case when $x_0 \in (\frac{c}{2}, c)$

Let $X = c - x_0$, where $x_1 + x_2 = X$, then from the above graph (from flow f_1 and f_2) we have

$$\begin{aligned} \frac{w_1}{\theta - \tau(l_0 + l_1)} &= x_1 \\ \frac{w_2}{\theta - \tau(l_0 + l_2)} &= x_2. \end{aligned} \tag{4.9}$$

Then adding the two equations above gives

$$\frac{w_1}{\theta - \tau(l_0 + l_1)} + \frac{w_2}{\theta - \tau(l_0 + l_2)} = X. \tag{4.10}$$

From the above we can solve for θ in term of X to have $\theta = g(X) = g(c - x_0)$. We only need to compare $g(c - x_0)$ with θ_0 which gives us the idea of binary search to find x_0 such that

$$\theta_0 = \frac{w_1 + w_2 + w_3}{x_0} + \tau l_0 = g(c - x_0). \tag{4.11}$$

With x_0 fixed, we can compare θ_0 with $g(c - x_0)$ and if $\theta_0 > g(c - x_0)$ then we can increase x_0 otherwise we decrease x_0. If $x_0 = X$ and $\theta_0 < g(c - x_0)$, then we have the optimal solution

already because we cannot increase or decrease x_0 anymore.

Case 2: Now suppose $x_0 \in (\frac{c}{3}, \frac{c}{2})$ and we still maintain $x_1 + x_2 = X$. We note that $X \in (\frac{c}{2}, \frac{2c}{3})$ and θ_1 dominates θ_2 and θ_{12}. This is because if θ_2 or θ_{12} dominates, x_1 can be reduce by $\varepsilon > 0$ capacity which increases x_2. We can continue to decrease x_1 until θ_1 dominates. We must also remember that the value assigned to x_1 and x_2 are limited by x_0. Let us examine the dynamic of the flow f_{12} in relation to the flow f_2 using the graph below

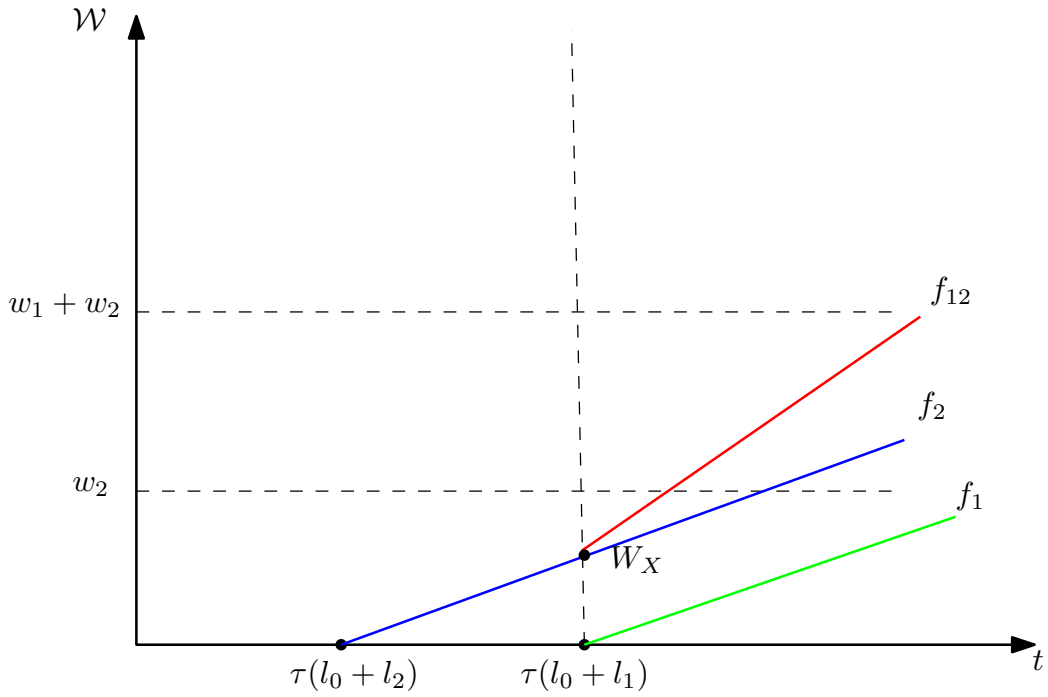

Figure 4.5: Graph illustrating the case when $x_0 \in (\frac{c}{3}, \frac{c}{2})$. Note that the slope of f_{12} is not smaller than the larger of x_1 and x_2 but limited by x_0.

The slope of f_{12} is not smaller than the larger of the slope of f_1 and f_2 but not greater than x_0. If we increase x_1 and decrease x_2 then θ_1 decreases while θ_2 increases and also θ_{12} increases because the parallel line f_2 decreases. Now if x_2 is increased to its upper bound $c - x_0$ then $\theta_{12} > \theta_2$, however, x_2 must not be greater than x_0. Geometrically we derive θ_{12} as follow: From the line f_2 and f_1 we have

$$\frac{w_1}{\tau(l_1 - l_2)} = x_2 \Longrightarrow w_1 = x_2\tau(l_1 - l_2). \tag{4.12}$$

Considering the starting point of line f_{12} and its slope we have

$$\frac{w_2 + w_1 - w_1}{\theta_{12} - \tau(l_0 + l_1)} = x_1 \Longrightarrow w_2 = \theta_{12}x_1 - x_1\tau(l_0 + l_1). \tag{4.13}$$

Adding Eq. (4.12) and Eq.(4.13) and solving for θ_{12} gives

$$\theta_{12} = \frac{w_1 + w_2 - x_2\tau(l_1 - l_2)}{x_1} + \tau(l_0 + l_1). \tag{4.14}$$

When x_2 is closer to zero, θ_2 dominates θ_{12} and therefore there is a point for x_2 where $\theta_2 = \theta_{12}$. Therefore if we can determine the point x_2 such that $\theta_{12} = \theta_2$, then we can compare with θ_1. If θ_1 is larger then we can decrease x_2 and from there we know that $\theta_2 > \theta_{12}$ and we can always compare θ_1 and θ_2. On the other hand if $\theta_1 < \theta_{12} = \theta_2$ at the critical point x_2, then we can increase x_2 and we know that $\theta_{12} > \theta_2$. Therefore we can compare θ_1 and θ_{12} to search for the optimal solution by binary search by comparing θ_1 and θ_{12}. When the best x_1 and x_2 are computed after the binary search, the maximum cost dictated by θ_1 and θ_2 or θ_2 and θ_{12} is compared with θ_0 using binary search.

Now let us look at the following graph to analyse the weight w_X such that $\theta_2 = \theta_{12}$ given x_0 fixed.

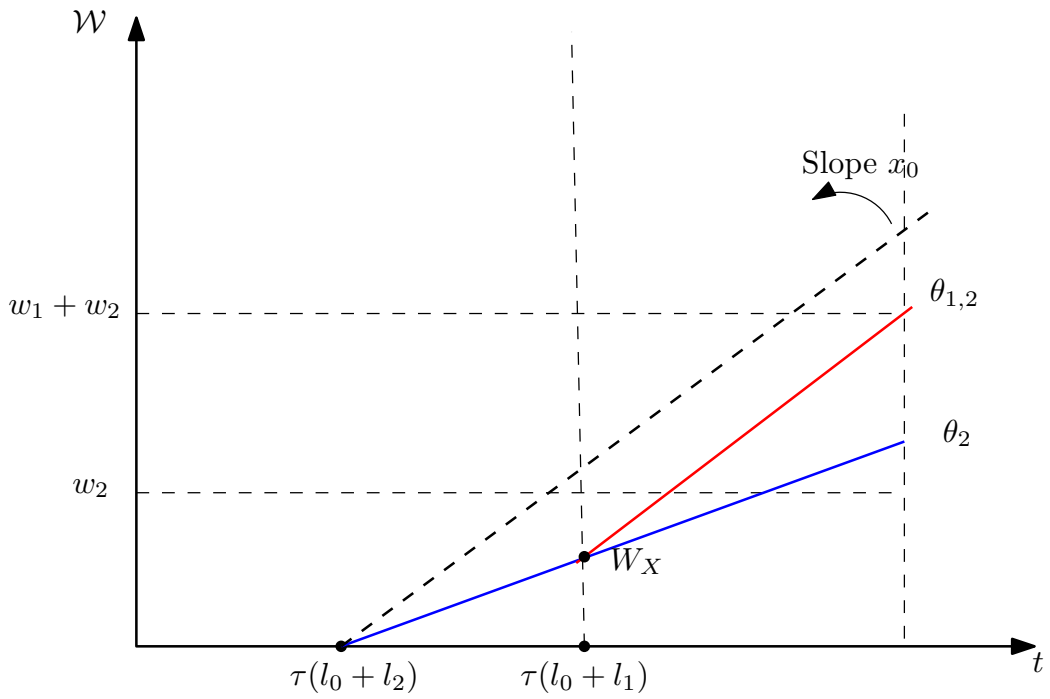

Figure 4.6: Graph showing the critical weight W_X at which $\theta_2 = \theta_{12}$.

Using the slope of f_2 together with the similar triangle in the graph we have

$$\frac{w_2}{\theta_2 - \tau(l_0+l_2)} = \frac{w_X}{\tau(l_1-l_2)}. \tag{4.15}$$

Using the line f_{12} we have

$$\frac{w_1+w_2-w_X}{\theta_2 - \tau(l_0+l_1)} = x_0. \tag{4.16}$$

With Eq. (4.15) and Eq. (4.16) we have a quadratic equation in θ_2 when w_X is eliminated:

$$\begin{aligned} &x_0\theta_2^2 - \theta_2(x_0\tau(l_0+l_2) + x_0\tau(l_0+l_1) + (w_1+w_2)) \\ &+ x_0\tau^2(l_0+l_1)(l_0+l_2) + (w_1+w_2)\tau(l_0+l_2) + w_1\tau(l_1-l_2) = 0. \end{aligned} \tag{4.17}$$

Once θ_2 is computed, x_2 can be computed and we know that $\theta_2 = \theta_{12}$ at the computed x_2. Then we can compare with θ_1 to know which direction to go. Thus the quadratic equation gives us the critical value of θ_2 (or x_2) before the comparison (by binary search) procedure.

The case of an extended star network with the sink located at a leaf node is more com-

plicated as there are many flow of the form f_{ij} whose influence in the set of optimal cost need to be tracked or considered.

Chapter 5

Minsum capacity provisioning on path networks

In this chapter, we turn attention to capacity provisioning problem on path networks, with the objective of minimizing the total evacuation time of all evacuees in the topology. We call this problem the minsum capacity provisioning problem in relation to the traditional minsum sink location problem [24, 37]. Recall that the minsum sink location is more difficult in any topology of interest compared to the minmax problem (see Section 1.2), and in fact the study of the problem is limited to path topology in literature (see the survey paper [25]). However, the minsum problems are more attractive when considering the psychological duress that may be suffered by the evacuees on their way to safe-haven.

As we have learnt in Chapter 3, the traditional idea of sink location algorithms is based on the possible evacuation by foot, which is effective in when considering dense communities. As for remote or sparse communities, evacuation procedure surely require some adjustments to the traditional model. Just as in in Chapter 3, we shall view the budgeted capacity as resource to be allocated optimally to the edges of the network in a way that the evacuation total time is minimized, and we require the sum of the allocated capacities to be no more than the budgeted capacity. The next section present the former definition of the problem, as well as the model formulation. We shall spend more time in the section to properly investigate and establish the problem with some examples.

5.1 Preliminaries and model formulation for the minsum capacity provisioning concept

Let $P = (V, E)$ represents a path graph with a vertex set $V = \{v_1, v_2, \ldots, v_{n+1}\}$ and an edge set $E = \{e_i = (v_i, v_{i+1}), i = 1, \ldots, n\}$. Each vertex v_i has a non-negative weight w_i which represents the number of evacuees situated at v_i, and each edge e_i has a fixed length l_i but unknown capacity x_i, which represents the upper limit on the number of evacuees that can be evacuated through the edge e_i in a unit time. The parameter c denotes a given budgeted capacity to be shared optimally between all the edge lengths and τ denotes the transit time per unit distance. We define the minsum capacity provisioning formerly as follow:

Problem 5.1. Given a dynamic path network $\mathcal{N} = (P, c, \tau, \mathbf{w})$, a fixed sink $S \in P$, and a total capacity budget c. Let $z(\mathbf{x})$ denote the sum of the evacuation times of all evacuees on P given the capacity vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ where x_i represents the capacity assigned to edge e_i on the path. Find the optimal capacity vector $\mathbf{x}^\star$ such that $z(\mathbf{x}^\star)$ is minimum and $\sum_{i=1}^{n} x_i = c$.

Let us investigate the problem on the following simple two-edged network (Figure 5.1), where the sink is fixed at the right end of the network and evacuees from v_1 and v_2 have to move at the same time to the sink. We let $x_2 = x$ so that $x_1 = c - x$.

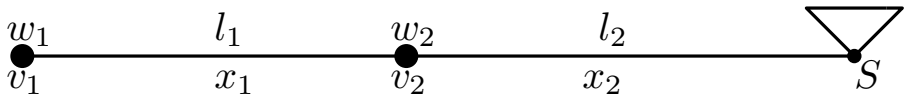

Figure 5.1: Illustration of capacities allocation on a path network with two edges. The sink is assumed to be located at the end vertex on the right.

There are two cases to consider:

[**Case 1 (No Congestion):**] Let us assume that there is no congestion. That is, the first set of evacuees from v_1 gets to v_2 and moves forward to the sink without any delay at v_2.

Mathematically, this implies that $\tau l_1 \geq \frac{w_2}{x}$ or $x \geq \frac{w_2}{\tau l_1}$. We can derive the objective function by adding the areas of the two trapeziums in Figure 5.2. Thus we have

$$z(x) = \frac{w_1^2}{2(c-x)} + \frac{w_2^2}{2x} + w_1\tau(l_1+l_2) + \tau l_2 w_2. \tag{5.1}$$

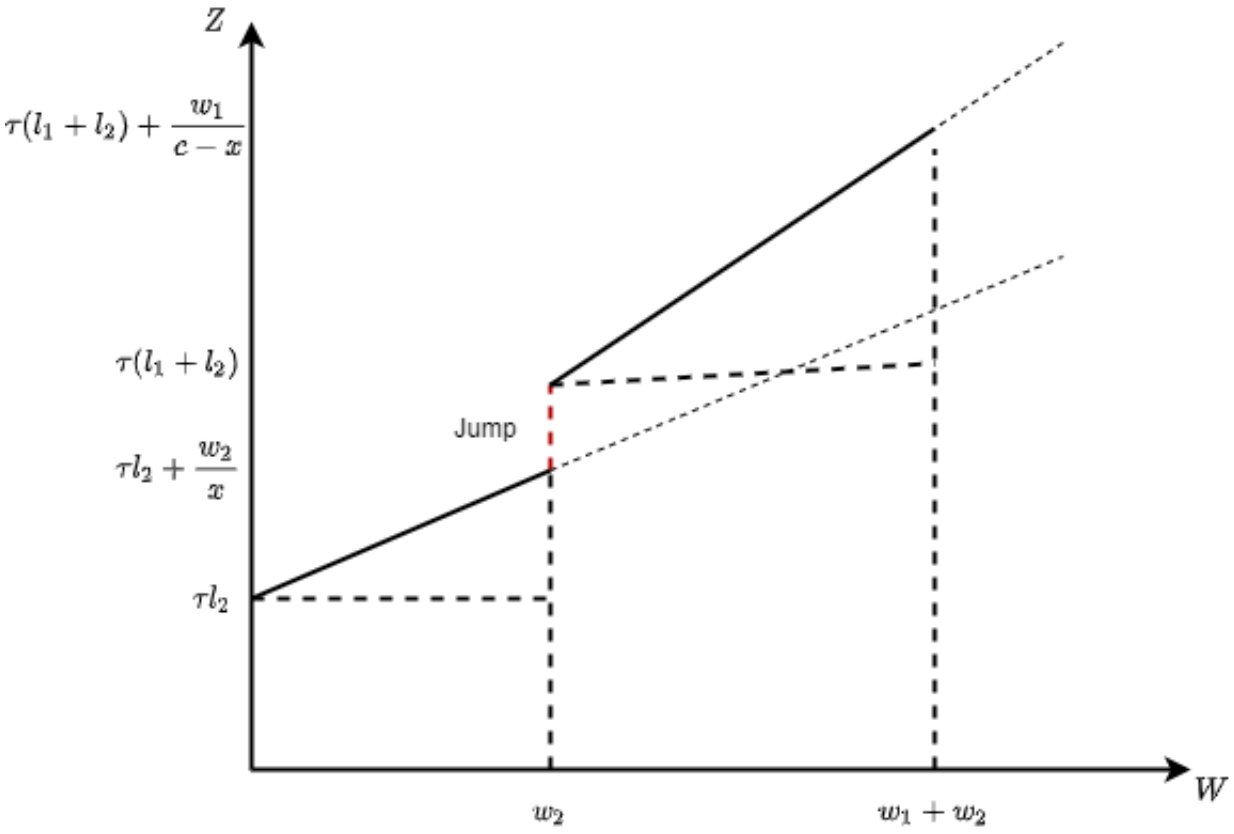

Figure 5.2: Time-weight graph of Figure 5.1. There is no congestion and evacuees get to the sink with their starting rates

From the objective function, if $x < \frac{c}{2}$ then the cost becomes

$$z(x) = \frac{w_1^2+w_2^2}{2x} + \tau l_2 w_2 + \tau(l_1+l_2)w_1, \tag{5.2}$$

which we can easily improve by allocating equal capacity to both edges. Thus trivially, the

cost

$$z(x) = \frac{w_1^2 + w_2^2}{c} + \tau l_2 w_2 + \tau(l_1 + l_2) w_1 \tag{5.3}$$

is better (lesser) than the cost in (5.2) since $2x < c$. Can we say we have reached the optimal cost and solution with the above? This will be clarified with the analysis below.

First, we observe that the objective function can be re-written as

$$z(x) = w_1 \tau(l_1 + l_2) + \tau l_2 w_2 + \min_{x \in [\frac{c}{2}, c)} g(x), \tag{5.4}$$

where

$$g(x) = \frac{w_1^2}{2(c-x)} + \frac{w_2^2}{2x}. \tag{5.5}$$

Since the first two terms of (5.4) are constant, only the function $g(x)$ is needed to be analysed. We also observe that the function $g(x)$ is convex for $x \in [c/2, c]$ and hence must have one local minimum (correspondingly global minimum in the specified interval). To check this, we use the second derivative test for the roots of $g'(x) = 0$ as follow. Computing $g'(x) = 0$ leads to the following quadratic equation

$$x^2(w_1^2 - w_2^2) + w_2^2 cx - w_2^2 c^2 = 0, \tag{5.6}$$

whose roots are

$$x_{a,b} = \frac{-2w_2^2 c}{2(w_1^2 - w_2^2)} \pm \sqrt{\frac{4w_2^4 c^2 + 4(w_1^2 - w_2^2) w_2^2 c^2}{4(w_1^2 - w_2^2)^2}}. \tag{5.7}$$

Further simplification yields

$$\left[x_a = \frac{-w_2 c}{w_1 - w_2},\ x_b = \frac{w_2 c}{w_1 + w_2} \right]. \tag{5.8}$$

Now using the second derivative test to examine the nature of each root derived, we first

discover that

$$g''(x) = \frac{w_1^2}{(c-x)^3} + \frac{w_2^2}{x^3} \tag{5.9}$$

is positive for any value of x in $[c/2, c)$. Now for the first root, we have

$$g''(x_a) = \frac{-(w_1 - w_2)^4}{c^3 w_1 w_2}, \tag{5.10}$$

which is obviously negative and hence x_a corresponds to a local maximum of $g(x)$. The second test gives

$$g''(x_b) = \frac{(w_1 + w_2)^4}{c^3 w_1 w_2}, \tag{5.11}$$

which is non-negative and therefore x_b corresponds to a local minimum. Furthermore, since $g(x)$ is convex, x_b also corresponds to its global minimum point (and global minimum of z) in the interval $[c/2, c]$.

Therefore for the problem at hand if $c \geq \frac{2w_2}{\tau l_1}$, the optimal solution is $\max\left\{\frac{c}{2}, \frac{w_2 c}{w_1+w_2}\right\}$.

[Case 2:] Let us now consider the case when $c < \frac{2w_2}{\tau l_1}$. In this case, the optimal solution may be in congestion case or out of congestion as we shall soon find out. First let us draw the graph of the scenario.

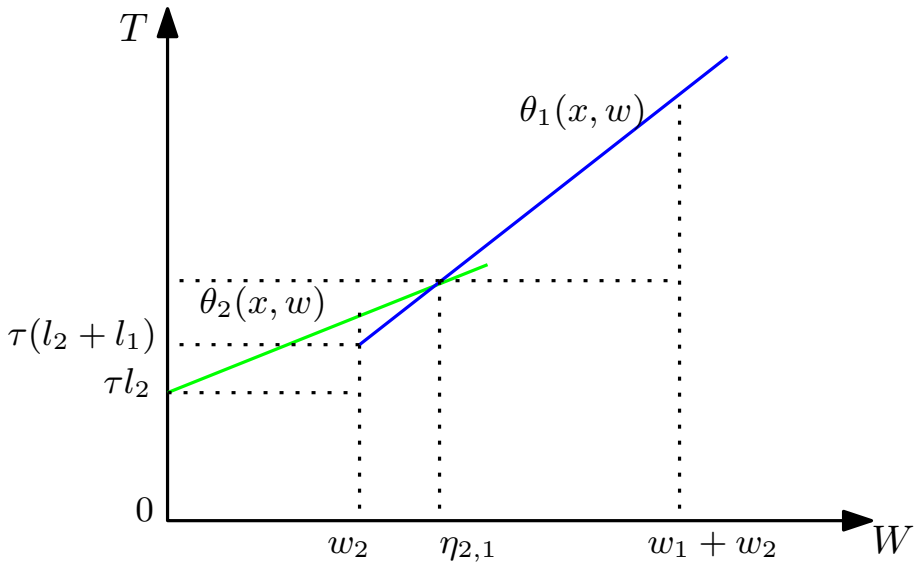

Figure 5.3: Congestion scenario for the path network in Figure 5.1. The intersection of the two straight lines indicate the time at which $\theta_2(x_2, \eta) = \theta_1(x_1, \eta)$

At $x = c/2$, the two lines θ_1 and θ_2 are parallel since they both have the same slope and the cost can be computed using the dominant of the two functions. However, the cost can be improved by increasing x to a point where the two lines intersect as shown in Figure 5.3. Moreover, there is a possibility that the optimal solution vector is in non-congestion configuration, which is case 1.From the graph we have the following time functions which represent the time for the last evacuees at v_2 and v_1 to get to the sink:

$$\begin{aligned} \theta_2(x,w) &= \tau l_2 + \frac{w}{x},\ w \geq 0 \\ \theta_1(x,w) &= \tau(l_1+l_2) + \frac{w-w_2}{\min\{c-x,x\}},\ w > w_2, \end{aligned} \tag{5.12}$$

however because of monotonicity property (which will be discussed later in Section 5.2) of the optimal solution and the geometric picture above, we define θ_1 as follow:

$$\theta_1(x,w) = \tau(l_1+l_2) + \frac{w-w_2}{c-x},\ w > w_2. \tag{5.13}$$

To find the value of η as a function of x, we need to minimize the following time

$$\Theta(x,\eta) = \max\{\theta_1(x,\eta), \theta_2(x,\eta)\}, \tag{5.14}$$

that is, we need to solve for η in

$$\tau(l_1+l_2) + \frac{\eta - w_2}{c-x} - \tau l_2 - \frac{\eta}{x} = 0. \tag{5.15}$$

Solving the above gives

$$\eta = \frac{w_2 x - \tau l_1 (c-x)x}{2x-c}. \tag{5.16}$$

Now equation (5.16) also verify that the two lines are parallel at $x = \frac{c}{2}$ as η does not exist at this point. Using Figure 5.3, we derive the objective function geometrically by adding the

areas of the two trapeziums featured in the graph. We have

$$z(x) = \frac{1}{2}(\theta_2(x,0) + \theta_2(x,\eta))\eta + \frac{1}{2}(\theta_1(x,\eta) + \theta_1(x, w_1 + w_2))(w_1 + w_2 - \eta), \tag{5.17}$$

where we have put $x_1 = c - x_2$ and $x_2 = x$. Expanding (5.17) with η defined in (5.16) gives

$$\begin{aligned} z(x) = \frac{1}{2}\frac{\left(2\tau l_2 + \frac{w_2 x - \tau l_1 (c-x)x}{(2x-c)x}\right)(w_2 x - \tau l_1 (c-x)x)}{2x-c} + \\ \frac{1}{2}\left(\tau l_2 + \frac{w_2 x - \tau l_1 (c-x)x}{(2x-c)x} + \tau(l_1 + l_2) + \frac{w_1}{c-x}\right) \times \\ \left(w_1 + w_2 - \frac{w_2 x - \tau l_1 (c-x)x}{2x-c}\right). \end{aligned} \tag{5.18}$$

Now we compute $z'(x) = 0$ to have

$$\begin{aligned} z'(x) = -2\tau^2 l_1^2 x^4 + 6c\tau^2 l_1^2 x^3 + (-7c^2\tau^2 l_1^2 + 2c\tau w_2 l_1 + 4w_1^2 - 2w_2^2)x^2 \\ + (4c^3\tau^2 l_1^2 - 4c^2\tau w_2 l_1 - 4cw_1^2 + 4cw_2^2)x - \\ - c^4\tau^2 l_1^2 + 2c^3\tau w_2 l_1 + c^2 w_1^2 - 2c^2 w_2^2 = 0. \end{aligned} \tag{5.19}$$

Compactly,

$$Ax^4 + Bx^3 + Cx^2 + Dx + E = 0, \tag{5.20}$$

where A, B, C, D, E are the coefficients in the polynomial (5.19).

The roots of (5.20) that lies in $(\frac{c}{2}, c)$ can only be found numerically as the closed form analytical method does not exist. However, we note that if $z(x)$ is convex in the interval $(c/2, x)$, then one of the roots of $z'(x)$ (which corresponds to the local minimum) lies in the interval.

Lemma 5.2. *Given a path network with two edges and a sink located at its end. The minsum capacity objective functions associated with the network for both congestion and non-congestion cases are convex.*

Proof. Note that monotonicity implies $x_2 \geq x_1$. The Hessian matrix associated with the non-congestion case is

$$H_{Non} = \begin{bmatrix} \frac{w_1^2}{x_1^3} & 0 \\ 0 & \frac{w_2^2}{x_2^3}. \end{bmatrix} \tag{5.21}$$

The determinant of $k \times k$, $k = 1,2$ submatrix of H_{Non} are non-negative, hence H is positive definite. For the congestion case, we have the determinant of its 1×1 matrix to be

$$\frac{l_1^2\tau^2 x_1^3 x_2^2 - 2l_1\tau w_2 x_1^3 x_2 - w_1^2 x_1^3 + 3w_1^2 x_1^2 x_2 - 3w_1^2 x_1 x_2^2 + w_1^2 x_2^3 + w_2^2 x_1^3}{x_1^3(x_2 - x_1)^3}. \tag{5.22}$$

The denominator is non-negative since $x_2 \geq x_1$. The numerator simplifies to

$$x_1^3(w_2 - l_1\tau x_2)^2 + w_1^2(x_2 - x_1)^3 \tag{5.23}$$

and also non-negative. The determinant of its 2 by 2 submatrix gives

$$\frac{w_1^2(w_2 - \tau l_1 x_1)^2}{x_1^3(x_2 - x_1)^3} \tag{5.24}$$

which is non-negative. Therefore the Hessian matrices associated with the two-edged network are positive definite. □

Let us further examine the problem with some examples.

Example 5.3. Let the budgeted capacity c be equal to 10, weights w_1 and w_2 be equal to 10 and 18 respectively. Let the transit time per unit length τ be equal to 1, and edge lengths l_1 and l_2 be 3 and 4 respectively. Clearly, $c < \frac{2w_2}{\tau l_1}$. If we directly assume that the optimal solution would be in congestion case, then the objective function computed from (5.17)

gives

$$\begin{aligned} z = \frac{1}{2}\frac{x(-12+3x)^2}{(2x-10)^2} + \frac{4x(-12+3x)}{2x-10} \\ + \frac{1}{2}\Big(28 - \frac{x(-12+3x)}{2x-10}\Big)\Big(11 + \frac{-12+3x}{2x-10} + \frac{10}{10-x}\Big). \end{aligned} \tag{5.25}$$

Now $z'(x) = 0$ gives

$$-18x^4 + 540x^3 - 5468x^2 + 23360x - 36800 = 0 \tag{5.26}$$

The zero of (5.26) that lies in $(5,10)$ is a single value $x = 6.2825$. The second derivative test shows that this value corresponds to a local minimum. The cost at this point is $z(6.2825) = 181.3187$. If the optimal solution is in a congestion scenario, then the value of η at this point must be between $w_2 = 18$ and $w_1 + w_2 = 28$. The value of η at this point gives 16.7760 which contradicts our analysis. In fact if the optimal solution lies in a congestion case as we have assumed, then this point must lie between $x_{\min}$ and $x_{\max}$ which correspond to when η is equal to $\eta_{\max} = w_1 + w_2$ and $\eta_{\min} = w_2$ respectively. To further verify this, we compute $x_{\min}$ as follow:

$$\begin{aligned} w_1 + w_2 &= \frac{w_2 x - \tau l_1 (c-x)x}{2x-c} \\ 28 &= \frac{18x - 3(10-x)x}{2x-10} \\ 0 &= 3x^2 - 68x + 280. \end{aligned} \tag{5.27}$$

Solution to the quadratic equation above is $x = 17.25$ or $x = 5.40$. Since the first solution is above the budgeted capacity, we can discard it, hence $x_{\min} = 5.40$. To compute the cost at $x_{\min}$, we need the following objective function

$$z(x_{\min}, \eta_{max}) = \tau l_2 \eta_{\max} + \frac{\eta_{\max}^2}{2x_{\min}} \tag{5.28}$$

which is derived when the two lines intersect at $w_1 + w_2$. Thus the cost at $x_{\min}$ gives $z(x_{\min}) = 184.5925$. In similar way, we can compute $x_{\max}$ as follow:

$$\begin{aligned} w_2 &= \frac{w_2 x - 3(c-x)x}{2x-c} \\ 18 &= \frac{18x - 30x + 3x^2}{2x-10} \\ 0 &= 3x^2 - 48x + 180. \end{aligned} \tag{5.29}$$

Solving the quadratic equation above gives $x = 10$ or $x = 6$. We discard the first as it is equal to the budgeted capacity and $x_{\max} = 6$. We can derive the corresponding objective function from Figure 5.4 by adding the areas of the two trapeziums in the figure. Thus we have

$$z(x_{\max}, \eta_{\min}) = \frac{1}{2}\left(2\tau l_2 + \frac{\eta_{\min}}{x_{\max}}\right) w_2 + \frac{1}{2}\left(2\tau l_2 + \tau l_1 + \frac{\eta_{\min}}{x_{\max}} + \frac{w_1}{c - x_{\max}}\right) w_1. \tag{5.30}$$

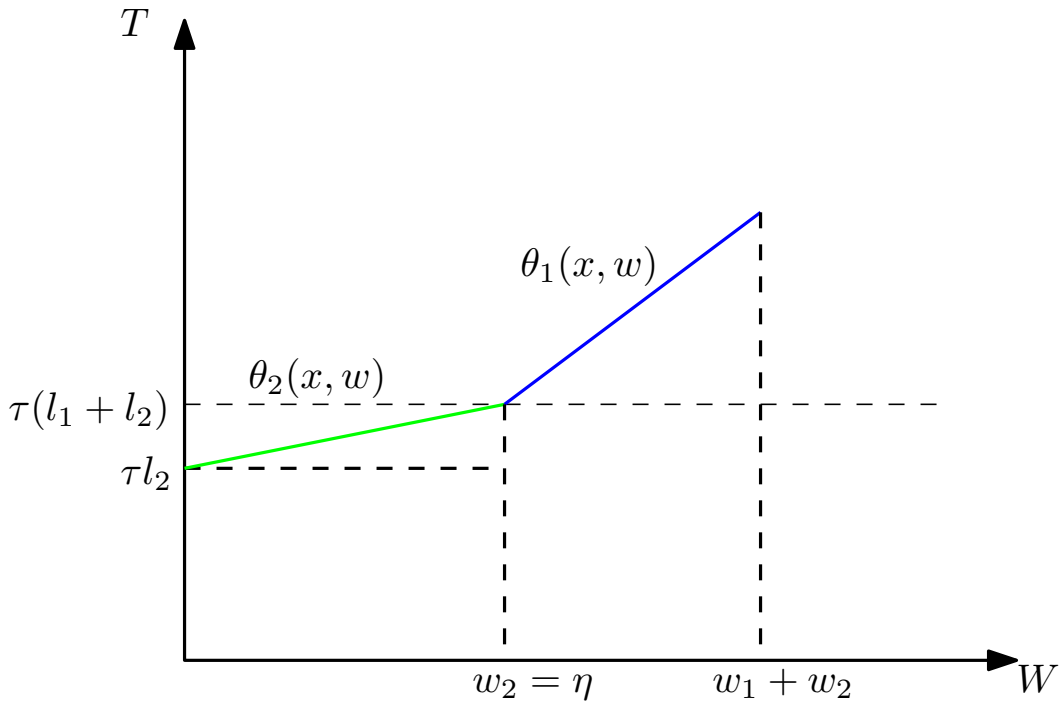

Figure 5.4: The case when $\eta = \eta_{\min}$ which corresponds to maximum value of x if the optimal solution is in congestion scenario

Now at $x_{\max} = 6$, $z(x_{\max}, w_2) = 181.50$. We discover that the cost derived when $x = 6.2825$ is better than the cost when x is equal to $x_{\min}$ and $x_{\max}$. However, this value is not between $x_{\min}$ and $x_{\max}$. Note that we have congestion when $x \in (5,6]$. Can we find a value of x in this interval that gives a better cost? This means we need to minimize

$$\begin{aligned} z(x) = &\frac{1}{2}\left(4+\frac{12-3x}{10-2x}+7+\frac{10}{10-x}\right)\left(28-\frac{12x-3x^2}{10-2x}\right) \\ &+\frac{1}{2}\left(4+4+\frac{12-3x}{10-2x}\right)\left(\frac{12x-3x^2}{10-2x}\right) \end{aligned} \tag{5.31}$$

with $x \in [5,6]$. Unfortunately the equation $Z'(x) = 0$ has no zero in $[5,6]$. Now as we increase x beyond 6, we move away from congestion and the line $\theta_1(x)$ goes beyond the line $\theta_2(x)$. The objective function in this case is (the no congestion case)

$$\begin{aligned} z(x) &= \frac{1}{2}(\tau l_2 + f_2(x, w_2))w_2 + \frac{1}{2}(\tau(l_1+l_2) + f_1(x, w_1+w_2))(w_1+w_2-w_2) \\ &= 142 + \frac{162}{x} + \frac{50}{10-x}. \end{aligned} \tag{5.32}$$

Minimizing the above function with $x \in [6,10]$ gives $x = 6.4287$ with the cost 181.20. Since this solution gives a better cost compared to others derived previously, we can write (the optimal solution vector) $\mathbf{x}^\star = [3.57, 6.43]$.

The above example shows that the initial situation or configuration may be a congestion scenario but the optimal solution may not. We can easily verify the no congestion scenario by checking the inequality $\tau l_1 > \frac{w_2}{x^\star}$. The next example shows that the optimal solution may remain in congestion configuration.

Example 5.4. Let $c = 20$, $\tau = 1$, $w_1 = 100$, $w_2 = 125$, $l_1 = 10$ and $l_2 = 8$. Then the objective function gives

$$z(x) = -\frac{25}{4}\frac{4x^3 - 708x^2 + 19215x - 125100}{(x-10)(-20+x)}. \tag{5.33}$$

Computing $z'(x) = 0$ leads to

$$-200x^4 + 12000x^3 - 221250x^2 + 1650000x - 4500000 = 0. \tag{5.34}$$

The zero of the polynomial (5.34) that lies in $(10,20)$ is derived uniquely (numerically) as $x = 11.7843$ with the cost $z(x = 11.784) = 4076.5527$. The value of η at this point is 141.4764. Since $w_2 < \eta(x) < w_1 + w_2$, the solution is in congestion case. To further verify that the above solution is the best, we compute $x_{\min} = 10.7884$ (corresponding to $\eta_{\max} = 225$) and $x_{\max} = 12.5$ (corresponding to $\eta_{\min} = 125$). Now $z(x_{\max}) = 4091.6666$ and $z(x_{\min}) = 4146.2801$. Since $x = 11.784$ gives the best cost, we write $x^\star = 11.784$. We do not need to search for solution in $[12.5,20]$ because the corresponding "no congestion" objective function has no zero in the specified interval.

We present the algorithm for the two-edged minsum capacity provisioning below. Note that from the analysis above, the candidate solution for x_2 are $x_{\text{half}} = \frac{c}{2}$, $x_b = \frac{w_2 c}{w_1+w_2}$, $x_max = \frac{w_2}{\tau l_1}$ and the root x_{num} of the order 4 polynomial (5.19) that lies in $(c/2, x_{\max})$.

Algorithm 7: Capacity allocation algorithm for two-edged network with sink located at its end.

Result: Output optimal cost and vector of edge capacities for the network in Fig. 5.1

Inputs: w_1, w_2, c, τ, l_1, l_2;

Compute the candidate vector for x_2 that is, $\bar{x}_2 = \{x_{\text{half}}, x_b, x_{\max}, x_{\text{num}}\}$;

Compute the corresponding x_1, that is $\bar{x}_1 = c \cdot \mathbf{1} - \bar{x}_2$;

Compute the corresponding values of objective functions $z(\mathbf{x})$;

Output the vector $\mathbf{x} = [x_1, x_2]$ that gives the minimum cost.

5.2 Properties of the optimal solution vector and cost

From the classical sink location problem we understand that the sink must be on a vertex (see [24] and Section 1.2), we repeat the property here as a lemma:

Lemma 5.5. *The optimal location of the sink s must be a vertex.*

Now for capacity allocation problem, we have the following properties:

Theorem 5.6. *Given a path P with $n+1$ vertices where $v_1, \dots v_n$ are occupied by evacuees $w_1, \dots, w_n$ and the sink is located at v_{n+1}. Let $\mathbf{x}^\star = (x_1^\star, \dots, x_n^\star)$ be the optimal edge-capacities vector and $z(\mathbf{x}^\star)$ denotes the total evacuation time, then $x_i^\star \geq x_j^\star$ if $i > j$.*

Proof. Let us first assume that the flow is congestion free. WLG let us consider flow from indexes v_i and v_{i+1} We have the following evacuation times at v_i and v_{i+1} respectively:

$$\begin{aligned} \theta_i &= \frac{w_i}{\min\{x_i, x_{i+1}\}} + \tau d(v_i, s) \\ \theta_{i+1} &= \frac{w_{i+1}}{x_{i+1}} + \tau d(v_{i+1}, s), \end{aligned} \tag{5.35}$$

and individual contributions to $z(\mathbf{x}^\star)$ gives

$$\begin{aligned} z^i &= \frac{w_i^2}{2\min\{x_{i+1}, x_i\}} + \tau d(v_i, s) w_i \\ z^{i+1} &= \frac{w_{i+1}^2}{2x_{i+1}} + \tau d(v_{i+1}, s) w_{i+1}. \end{aligned} \tag{5.36}$$

Since we can reduce z^{i+1} (or overall reduce $z(\mathbf{x}^\star)$) by making $x_{i+1} \geq x_i$ then $x_{i+1}^\star > x_i^\star$ in $\mathbf{x}^\star$.

Now suppose there is a congestion at v_{i+1}, where some evacuees from v_i encounter a delay at v_{i+1} due to backlog of evacuees waiting to depart v_{i+1}, then there exist some supply η_{i+1} that constitutes part of supply from v_i that merge with some supply at v_{i+1}. Therefore we have the following time functions:

$$\begin{aligned} \theta_i &= \frac{w_i}{\min\{x_{i+1}, x_i\}} + \tau d(v_i, s) \\ \theta_{i+1} &= \frac{\eta_{i+1}}{x_{i+1}} + \tau d(v_{i+1}, s), \end{aligned} \tag{5.37}$$

and contribution to $z(\mathbf{x}^\star)$ gives

$$\begin{aligned} z^{i,i+1} &= \frac{\eta_{i+1}^2}{2x_{i+1}} + \tau d(v_{i+1},s)\eta_{i+1} \\ &+ \frac{1}{2}\left(\tau l_i + 2\tau d(v_{i+1},s) + \frac{\eta_{i+1}}{x_{i+1}} + \frac{w_i}{\min\{x_{i+1},x_i\}}\right)(w_{i+1}+w_i-\eta_{i+1}) \end{aligned} \tag{5.38}$$

Obviously the above contributions can be reduced by making x_{i+1} greater than x_i in $\mathbf{x}^\star$. □

Theorem 5.7. *The minsum objective function z for the capacity provisioning problem is convex.*

Proof. Since the components of z can be broken down to set of convex functions, therefore their addition is convex. □

5.3 Extension to longer path networks

We have seen in the above section that the objective function for the minsum capacity provisioning is convex. However, it configuration could change during the optimization process. For simplicity, we shall focus on the objective function that defines the flow to the right $z_L\mathbf{x}$ and assume that the objective function for flow to the left is zero, i.e., $z_R(\mathbf{x}) = 0$. The flow to the right can be analysed symmetrically. Thus, if the sink is located on any interior vertex of the path, then

$$z(\mathbf{x}) = z_L(\mathbf{x}) + z_R(\mathbf{x}). \tag{5.39}$$

We also define for $j > i$, $W[i,j] = \sum_{k=i}^{j} w_k$, $L[i,j] = \sum_{k=i}^{j} l_k$ and for $j \geq 1$, $W[j] = \sum_{k=1}^{j} w_k$ and $L[j] = \sum_{k=1}^{j} l_j$. Let us consider the three edged network shown below:

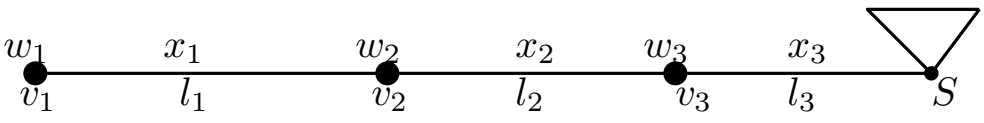

Figure 5.5: Path network with three edges and sink located on the right end.

Four possible initial configurations for z can be derived for this network:

Case 1: When we have congestion free scenario, that is, $\eta_{1,2} \leq w_3$ and $\eta_{2,1} \leq w_3 + w_2$, then the objective function is

$$z(\mathbf{x}) = \frac{1}{2}(\tau l_3 + \theta_3(x_3, w_3))w_3 + \frac{1}{2}(\tau L[2,3] + \theta_2(x_2, W[2,3]))w_2 + \frac{1}{2}(\tau L[1,3] + \theta_1(x_1, W[1,3]))w_1. \tag{5.40}$$

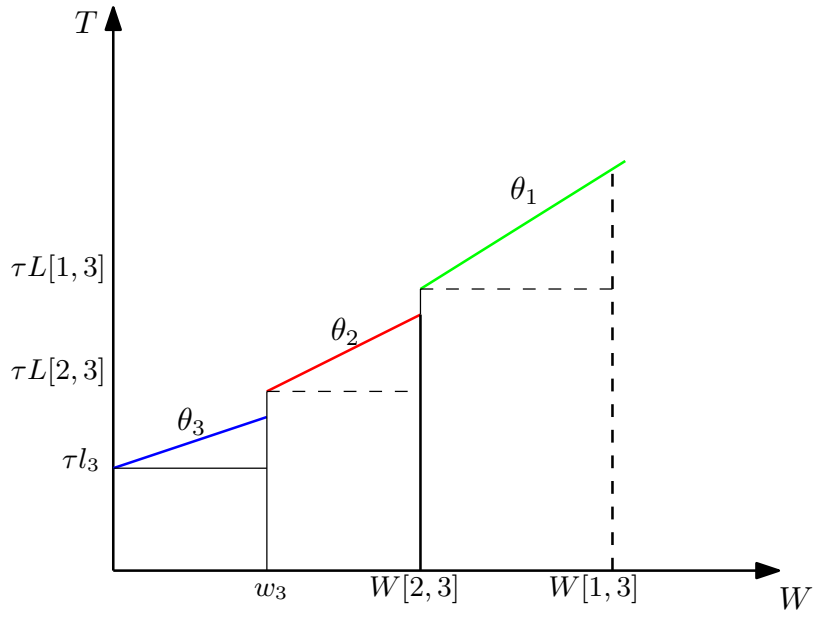

Figure 5.6: Congestion free scenario where $\eta_{3,2} \leq w_3$ and $\eta_{2,1} \leq W[2,3]$

Case 2: When we have Congestion only at v_3 but not at v_2. That is, $w_3 < \eta_{3,2} \leq W[2,3]$ and $\eta_{2,1} \leq W[2,3]$.

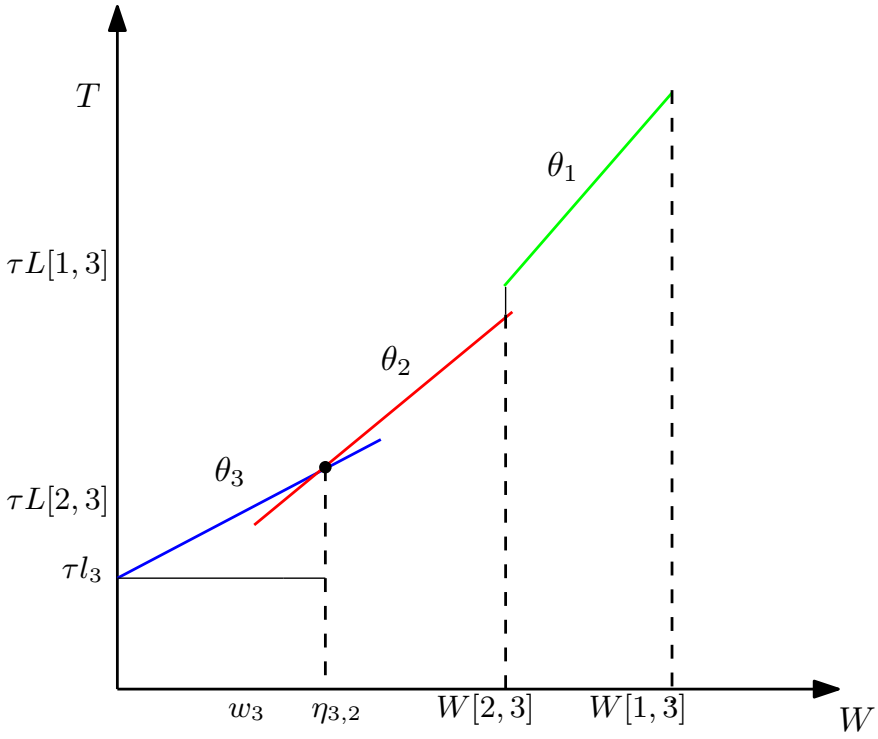

Figure 5.7: Congestion only at v_3

Note that if $\eta_{3,2} < W[2,3]$, then we have

$$z(\mathbf{x}) = \frac{1}{2}(\tau l_3 + \theta_3(x_3, \eta_{3,2}))\eta_{3,2} + \frac{1}{2}(\theta_2(x_2, \eta_{3,2}) + \theta_2(x_2, W[2,3]))(W[2,3] - \eta_{3,2}) + \frac{1}{2}(\tau L[1,3] + \theta_1(x_1, W[1,3]))w_1. \tag{5.41}$$

When $\eta_{3,2} = W[2,3]$ and $\eta_{2,1} \leq W[2,3]$, then

$$z(\mathbf{x}) = \frac{1}{2}(\tau l_3 + \theta_3(x_3, \eta_{3,2}))\eta_{3,2} + \frac{1}{2}(\tau L[1,3]) + \theta_1(x_1, W[1,3]))w_1, \tag{5.42}$$

which is covered by (5.41).

Case 3: When we have congestion only at v_2 and not at v_3. That is, $W[2,3] < \eta_{2,1} \leq W[1,3]$ and $\eta_{3,2} \leq w_3$. If $\eta_{2,1} < W[1,3]$, then

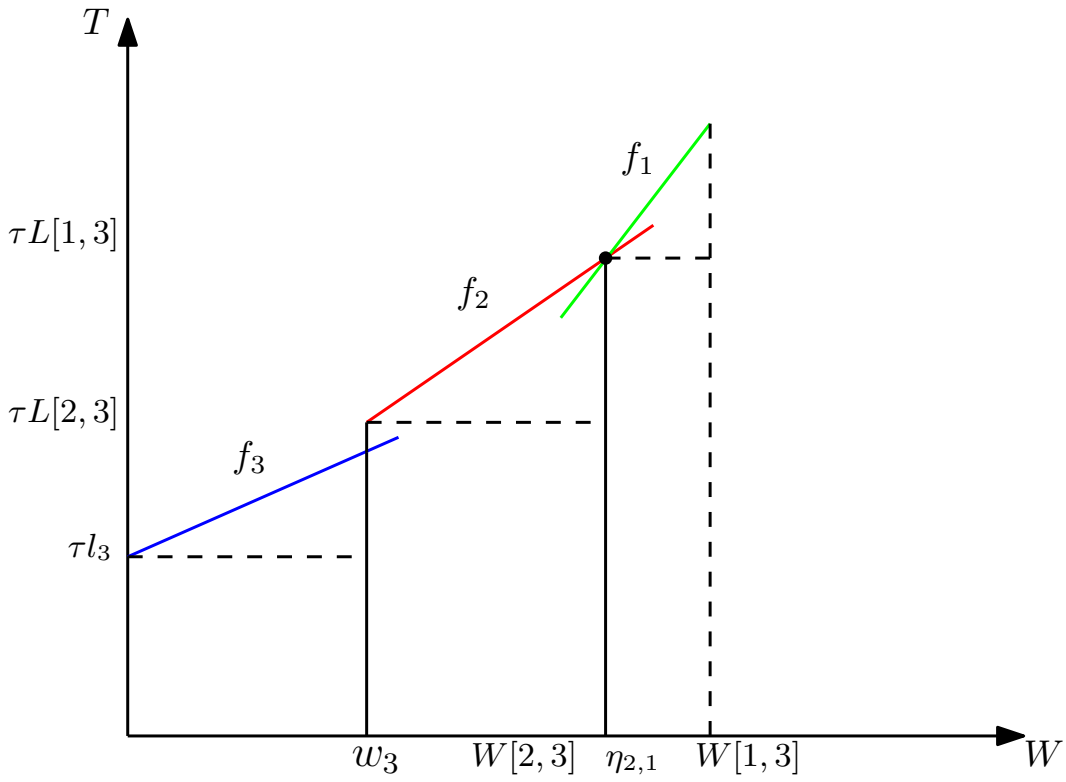

Figure 5.8: Congestion only at v_2

$$z(\mathbf{x}) = \frac{1}{2}(\tau l_3 + \theta_3(x_3, w_3))w_3 + \frac{1}{2}(\tau L[2,3] + \theta_2(x_2, \eta_{2,1}))(\eta_{2,1} - w_3) + \frac{1}{2}(\theta_2(x_2, \eta_{2,1}) + \theta_1(x_1, W[1,3]))(W[1,3] - \eta_{2,1}). \tag{5.43}$$

When $\eta_{2,1} = W[1,3]$, then

$$z(\mathbf{x}) = \frac{1}{2}(\tau l_3 + \theta_3(x_3, w_3))w_3 + \frac{1}{2}(\tau L[2,3] + \theta_2(x_2, \eta_{2,1}))(\eta_{2,1} - w_3), \tag{5.44}$$

which is covered by (5.43).

Case 4: If there is congestion at v_3 and v_2. That is, $w_3 < \eta_{3,2} \leq W[2,3]$ and $W[2,3] < \eta_{2,1} \leq W[1,3]$. Then we have

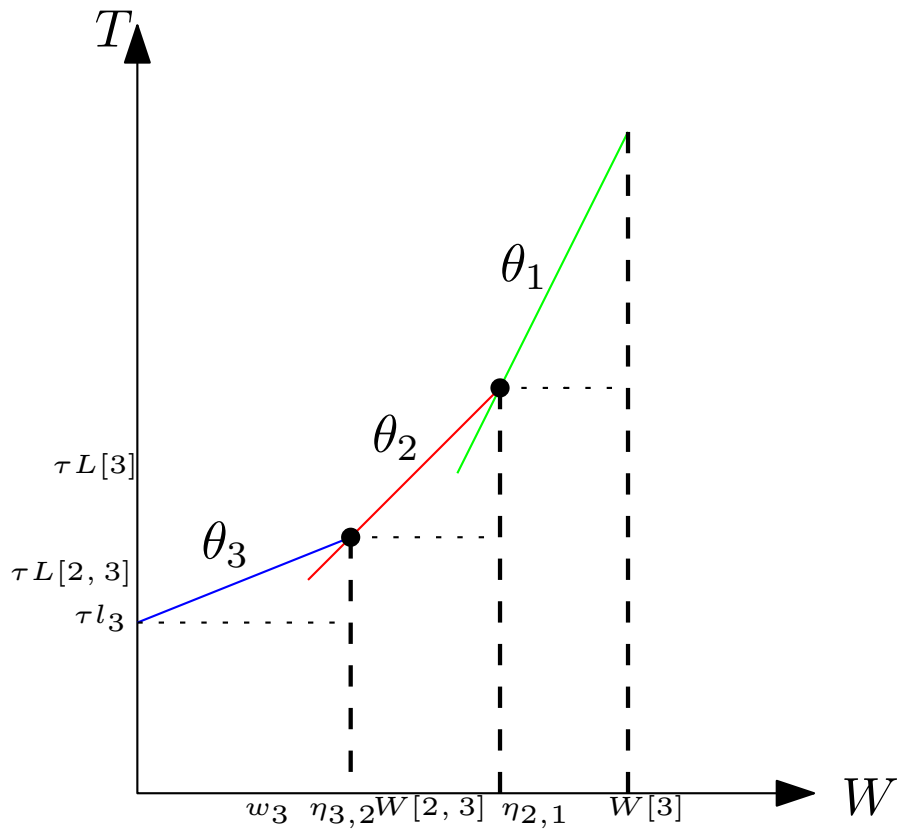

Figure 5.9: Congestion at both vertices

$$z(\mathbf{x}) = \frac{1}{2}(\tau l_3 + \theta_3(x_3, \eta_{3,2}))\eta_{3,2} + \frac{1}{2}(\theta_3(x_3, \eta_{3,2}) + \theta_2(x_2, \eta_{2,1}))(\eta_{2,1} - \eta_{3,2}) + \frac{1}{2}(\theta_2(x_2, \eta_{2,1}) + \theta_1(x_1, W[3]))(W[3] - \eta_{2,1}). \tag{5.45}$$

The above objective function also covers the cases $\eta_{3,2} = W[2,3]$, $\eta_{2,1} = W[3]$.

Therefore in general, given any path network of $n+1$ vertices where the sink is located on $(n+1)$th vertex, the minsum objective function for the capacity provisioning can be

written as:

$$z(\mathbf{x},\boldsymbol{\beta}) = \frac{1}{2}(\theta_n(x_n,0)+\theta_n(x_n,\beta_{n,n-1}))\beta_{n,n-1} + \frac{1}{2}(\theta_{n-1}(x_{n-1},\beta_{n,n-1})+\theta_{n-1}(x_{n-1},\beta_{n-1,n-2}))(\beta_{n-1,n-2}-\beta_{n,n-1}) + \cdots + \frac{1}{2}(\theta_2(x_2,\beta_{3,2})+\theta_2(x_2,\beta_{2,1}))(\beta_{3,2}-\beta_{2,1}) + \frac{1}{2}(\theta_1(x_1,\beta_{2,1})+\theta_1(x_1,W[n]))(W[n]-\beta_{2,1}), \tag{5.46}$$

where

$$\beta_{n,n-1} = \begin{cases} w_n & \text{if } \tau ln - 1 \geq \frac{w_n}{x_n} \\ \eta_{n,n-1}, & \text{otherwise,} \end{cases} \tag{5.47}$$

and for $2 \leq j < n$, we have

$$\beta_{j,j-1} = \begin{cases} W[j,n] & \text{if } \tau l_{j-1} \geq \frac{w_j}{x_j} \\ \eta_{j,j-1}, & \text{otherwise.} \end{cases} \tag{5.48}$$

Note again that $\eta_{j,j-1}$ is computed by solving for the weight in $\theta_j(x_j,w)-\theta_{j-1}(x_{j-1},w) = 0$, for $2 \leq j \leq n$. With the objective function defined, we need an algorithm that can accommodate the changing nature of the function given an initial configuration. We also need the algorithm to seamlessly accommodate the associated constraints, which include the positive capacity allocation, monotone capacity allocation, and the sum of the allocated capacity must be equal to the given budget c. We discuss our approach in the next section.

5.4 Solution by sequential quadratic programming

The sequential quadratic programming SQP is an iterative numerical method often employed to solve difficult optimization problems with constraints [19, 16]. The method incorporates the idea of Lagrange multiplier method to incorporate the constraints and then uses the idea of quadratic programming to make the numerical method faster (see Chapter 2). One important advantage of the SQP is that the initial solution vector can be made arbitrary.

However, because we understand that the optimal solution vector must be monotone, we incorporate a monotone initial solution vector into the SQP methods.

For our problem, we focus on the flow from left to right $z_L(\mathbf{x})$ and assume $z_R(\mathbf{x}) = 0$. We redefine the problem as follow:

$$\begin{aligned} \min \quad & z(\mathbf{x}) \\ \textbf{S.t.} \quad & x_1 + x_2 + \cdots + x_n = c \qquad (5.49) \\ & x_i - x_{i+1} \leq 0, \quad \forall 1 \leq i \leq n-1. \qquad (5.50) \end{aligned}$$

5.4.1 The SQP model and algorithm formulation

We employed the Newton SQP discussed in Chapter 2 . The Lagrangian objective function which incorporate the constraints with the objective function is given by

$$\mathcal{L} = z(\mathbf{x}) + \lambda_0(x_1 + x_2 + \cdots + x_n - c) + \sum_{j=1}^{C_{ineq}} \lambda_j (x_i - x_{i+1} + s_j^2), \qquad (5.51)$$

where s_j $1 \leq j \leq C_{ineq}$, ($C_{ineq} = 1 + (|E| - 2)$) are slack variables added to convert the inequality constraints to equality constraints, λ_0 is the Lagrange multiplier for the equality constraint in Eq.(5.49) and λ_j, $1 \leq j \leq C_{ineq}$ are the Lagrange multipliers for the converted inequality constraints.

Let $\boldsymbol{\Delta x} = (\mathbf{x} - \mathbf{x_k}, \mathbf{s} - \mathbf{s_k})'$ and let $\mathbf{g}$ be the vector of constraints, then the Taylor expansion of the Lagrangian function at the optimal solution gives

$$\mathcal{L}(\mathbf{x}^\star, \boldsymbol{\lambda}^\star) + \nabla \mathcal{L}(\mathbf{x}^\star, \boldsymbol{\lambda}^\star) \boldsymbol{\Delta x} + \frac{1}{2} \boldsymbol{\Delta x}' B(\mathbf{x}^\star, \lambda^\star) \boldsymbol{\Delta x}, \qquad (5.52)$$

where we have included the Lagrangian multiplier λ_0 in $\boldsymbol{\lambda}$ and B denotes the approximation of the Hessian of the Lagrangian $\mathcal{L}$. Now, linearizing the constraint vector at iterate $\mathbf{x}_k$ gives

$$\mathbf{g}(\mathbf{x}_k) + \nabla \mathbf{g}(\mathbf{x}_k)' \boldsymbol{\Delta x} + s_k + \Delta = \mathbf{0}. \qquad (5.53)$$

We now have the quadratic subproblem (see Chapter 2)

$$\min_{\Delta x} \quad \nabla \mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k)\boldsymbol{\Delta x} + \frac{1}{2}\boldsymbol{\Delta x}' B(\mathbf{x}_k, \boldsymbol{\lambda}_k)\boldsymbol{\Delta x} \tag{5.54}$$

$$\textbf{S.t.} \quad \nabla \mathbf{g}(\mathbf{x}_k)'\boldsymbol{\Delta x} + \nabla \mathbf{g}(\mathbf{x}_k) = 0. \tag{5.55}$$

The iterative method to be derive follows the same approach as the Newton's method as we shall see. Recall that the first order condition condition stipulates that $\nabla \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = 0$ (see Subsection 2.5.1). Differentiating the objective function in Eq.(5.54) with respect to $\boldsymbol{\Delta x}$ gives

$$\nabla z + \nabla \mathbf{g}(\mathbf{x}_k)\boldsymbol{\lambda}^\star + \boldsymbol{\Delta x}' B_k = 0. \tag{5.56}$$

The above equation can be rearranged to have

$$\boldsymbol{\lambda}^{\star'} \nabla \mathbf{g}(\mathbf{x}_k) + B_k \boldsymbol{\Delta x} = -\nabla z. \tag{5.57}$$

Setting $\boldsymbol{\lambda}^\star = \boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \boldsymbol{\Delta\lambda}$ and substituting in Eq.(5.57) give

$$\begin{aligned} -\nabla z - \boldsymbol{\lambda}_k' \nabla \mathbf{g}(\mathbf{x}_k) &= \boldsymbol{\Delta\lambda}' \nabla \mathbf{g}(\mathbf{x}_k) + B_k \boldsymbol{\Delta x} \\ -\nabla \mathcal{L}(\mathbf{x}_k, \boldsymbol{\Delta}_k) &= \boldsymbol{\Delta\lambda}' \nabla \mathbf{g}(\mathbf{x}_k) + B_k \boldsymbol{\Delta x}. \end{aligned} \tag{5.58}$$

Using Eq.(5.58) and the linearized constraint vector Eq.(5.53), we have the system

$$\begin{aligned} -\nabla \mathcal{L}(\mathbf{x}_k, \boldsymbol{\Delta}_k) &= \boldsymbol{\Delta\lambda}' \nabla \mathbf{g}(\mathbf{x}_k) + B_k \boldsymbol{\Delta x} \\ -\mathbf{g}(\mathbf{x}_k) &= \nabla \mathbf{g}(\mathbf{x}_k)' \boldsymbol{\Delta x}, \end{aligned} \tag{5.59}$$

which can be arranged in matrix form as

$$\begin{bmatrix} B(\mathbf{x}_k, \boldsymbol{\lambda}_k) & J_g^T(\mathbf{x}_k) \\ J_g(\mathbf{x}_k) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x} \\ \boldsymbol{\Delta\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla z(\mathbf{x}_k) + \boldsymbol{\lambda}_k' \nabla \mathbf{g}(\mathbf{x}_k) \\ \mathbf{g}(\mathbf{x}_k) \end{bmatrix} \tag{5.60}$$

and compactly as

$$J_{\mathcal{L}} \begin{bmatrix} \boldsymbol{\Delta x} \\ \boldsymbol{\Delta \lambda} \end{bmatrix} = -\nabla_{\mathbf{x},\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}), \tag{5.61}$$

where J_g denotes the Jacobian of the constraint vector $\mathbf{g}$ and $\nabla_{\mathbf{x},\boldsymbol{\lambda}}$ denotes the Jacobian of $\mathcal{L}$ with respect to $\mathbf{x}$ and $\boldsymbol{\lambda}$. The matrix on the right is the KKT or Jacobian matrix for the problem. The one-step iterates are then computed using:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{\Delta x} \\ \boldsymbol{\lambda}_{k+1} &= \boldsymbol{\lambda}_k + \boldsymbol{\Delta \lambda}, \end{aligned} \tag{5.62}$$

where we have merged the solutions for the slack variables to $\mathbf{x}_{k+1}$. That is $\mathbf{x}_{k+1} = [\mathbf{x}_{k+1}, \mathbf{s}_{k+1}]$.

We choose the initial Lagrangian multiplier vector $\boldsymbol{\lambda}_0$ to follow the first order optimality conditions or Karush-Kuhn-Tucker (KKT) conditions (see Eq.(2.54)):

$$\boldsymbol{\lambda}_0 = [J_g(\mathbf{x_0}, \mathbf{s_0}) J_g'(\mathbf{x_0}, \mathbf{s_0})]^{-1} (J_g(\mathbf{x_0}, \mathbf{s_0}) J_z'(\mathbf{x_0}, \mathbf{s_0})), \tag{5.63}$$

and the initial slack variable vector to be a set of ones. More importantly, we define our merit function to be the augmented function

$$\phi(\mathbf{x}, \boldsymbol{\gamma}) = z(\mathbf{x}) + \mathbf{g}(\mathbf{x})\boldsymbol{\lambda}' + \frac{\gamma}{2} \|\boldsymbol{g}(\boldsymbol{x})\|_2^2 \tag{5.64}$$

(where $\boldsymbol{\gamma}$ is a positive real number) to track the convergence of the objective function. Note that as the configuration of z changes, the merit function also changes too. The algorithm follows a Newton optimization idea.

Algorithm 8: Capacity allocation algorithm for an arbitrary path network in min-sum criterion

Result: Output the optimal cost and vector of edge capacities for the minsum problem

Inputs: $\mathbf{W}$, $\mathbf{L}$, c, τ;

Allocate equal capacities to determine the initial z configuration;

Construct the associated Lagrangian function $\mathcal{L}(\mathbf{x},\mathbf{s},\lambda)$;

Compute the Jacobian matrix $J_{\mathcal{L}}$ of $\mathcal{L}$;

Compute the right hand side vector $\nabla_{\mathbf{x},\boldsymbol{\lambda}}\mathcal{L}$;

Initialize a monotone initial condition;

while $\nabla\mathcal{L}(\mathbf{y}) > \varepsilon$ **do**

  Compute $\Delta\mathbf{x} = -J_{\mathcal{L}}^{-1} \times \nabla\mathcal{L}$;

  Compute the next solution $\mathbf{y_{i+1}} = \mathbf{y_i} + \Delta\mathbf{x}$;

  Check if $z(\mathbf{y_i})$ has changed the configuration and compute the new z function ;

end

Stop iteration when tolerance is met.

From the theory of quadratic programming problem (see [33]), we understand that when the number of constraints m is greater than or equal to 1, the KKT matrix in (5.60) is indefinite. To circumvent this problem and have effective direct method, we employ a symmetric indefinite factorization to decompose the KKT matrix. This factorization in this context is given by

$$P' J_{\mathcal{L}} P = L_d B_d L_d', \tag{5.65}$$

where P is a permutation matrix, L_d is a unit lower triangular matrix, and B_d is a block diagonal matrix with either 1×1 or 2×2 blocks. This allows us to solve for $[\Delta x\ \Delta\lambda]'$ systematically as follows: Solve

$$L_d \Delta\mathbf{y} = -P'\nabla_{\mathbf{x},\boldsymbol{\lambda}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) \tag{5.66}$$

for unknown $\Delta\mathbf{y}$, solve

$$B_d\hat{\Delta\mathbf{y}} = \Delta\mathbf{y} \tag{5.67}$$

for unknown $\hat{\Delta\mathbf{y}}$, and solve

$$L_d'\overline{\Delta\mathbf{y}} = \hat{\Delta\mathbf{y}} \tag{5.68}$$

for unknown $\overline{\Delta\mathbf{y}}$. Finally, set

$$\begin{bmatrix} \Delta\mathbf{x} \\ \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = P\hat{\Delta\mathbf{y}}. \tag{5.69}$$

Also for some specific problems, if $B(\mathbf{x},\boldsymbol{\lambda})$ in (5.60) is positive definite and well conditioned, then the Schur-Complement method can be employed (see [33] for details). However, $B(\mathbf{x},\boldsymbol{\lambda})$ cannot be assumed to be positive definite in general.

The algorithm 8 begins by allocating equal capacities to construct initial z configuration for the problem. It then goes further to construct the associated Lagrangian function. Due to the possible congestion parameter η, the algorithm employs initial solution vector that has no equal elements while ensuring a monotone allocation towards the sink. The algorithm also incorporates a template to monitor the change in the configuration of z at each iteration.

5.4.2 The Hessian approximation

The basic Newton SQP may encounter convergence difficulty if the initial solution $\mathbf{x_0}$ is chosen too far from the true solution. Moreover, when $\mathbf{x_0}$ is far from $\mathbf{x}^\star$, the sequence of Hessian approximations of the Lagrangian may not be positive definite on the required subspace which may hinder the existence of solutions of the quadratic subproblems. Based on the above reasons, the class of Hessian approximations discussed in Subsection 2.5.6 and Subsection 2.5.7 are employed. This class of approximations is referred to as secant approximation and has been found efficient for unconstrained non-linear problems. Basically, the Hessian approximations satisfy the secant equation, which is a version of bounded deterioration property and have some attractive convergence properties. We describe the se-

cant approximation below. Recall from Taylor's theorem that if the objective function $\mathcal{L}$ is continuously differentiable and given $\mathbf{p} \in \mathbb{R}^n$, then we have (see for example [33])

$$\mathcal{L}(\mathbf{x}+\mathbf{p}) = \mathcal{L}(\mathbf{x}) + \nabla\mathcal{L}(\mathbf{x}+t\mathbf{p})'\mathbf{p}, \tag{5.70}$$

for some $t \in (0,1)$. And if $\mathcal{L}$ is twice differentiable and continuous, we have

$$\nabla\mathcal{L}(\mathbf{x}+\mathbf{p}) = \nabla\mathcal{L}(\mathbf{x}) + \int_0^1 \nabla^2\mathcal{L}(\mathbf{x}+t\mathbf{p})\mathbf{p}\mathrm{dt}. \tag{5.71}$$

Now, adding and subtracting $\nabla^2\mathcal{L}(\mathbf{x})\mathbf{p}$ in Eq.(5.71), we have

$$\nabla\mathcal{L}(\mathbf{x}+\mathbf{p}) = \nabla\mathcal{L}(\mathbf{x}) + \nabla^2\mathcal{L}(\mathbf{x})\mathbf{p} + \int_0^1 [\nabla^2\mathcal{L}(\mathbf{x}+\mathbf{p}) - \nabla^2\mathcal{L}(\mathbf{x})]\mathbf{p}\mathrm{dt}. \tag{5.72}$$

The above equation reduces to

$$\nabla\mathcal{L}(\mathbf{x}+\mathbf{p}) = \nabla\mathcal{L}(\mathbf{x}) + \nabla^2\mathcal{L}(\mathbf{x})\mathbf{p} + O(\|p\|) \tag{5.73}$$

due to the continuity of $\nabla\mathcal{L}$. Letting $\mathbf{x} = \mathbf{x}_k$ and $\mathbf{p} = \mathbf{x}_{k+1} - \mathbf{x}_k$, we have

$$\nabla\mathcal{L}(\mathbf{x}_{k+1}) = \nabla\mathcal{L}(\mathbf{x}_k) + \nabla^2\mathcal{L}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) + O(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|), \tag{5.74}$$

and with the assumption that the iterates are close to the solution, we have the secant equation

$$\nabla\mathcal{L}(\mathbf{x}_{k+1}) - \nabla\mathcal{L}(\mathbf{x}_k) \simeq \nabla^2\mathcal{L}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{5.75}$$

Note that $\nabla^2\mathcal{L} = H_{\mathcal{L}}$ the Hessian of the Lagrangian. The secant approximations now find B_k that satisfies

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla\mathcal{L}(\mathbf{x}_{k+1}, \boldsymbol{\lambda}_{k+1}) - \nabla\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_{k+1}). \tag{5.76}$$

5.4.3 Numerical experiment

We show the results of our algorithm on 2-edge length, 3-edge length and 4-edge length path networks. We discovered that the results satisfy the monotone allocation condition even when the sink is located in the middle. The last rows of the tables present the optimal costs and solution vectors.

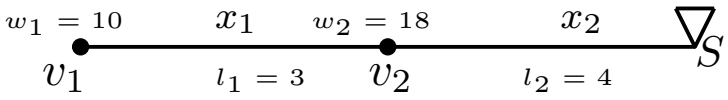

Figure 5.10: Path network with two edges.

z-value	x_1	x_2
161.65	6.6667	13.333
161.62	7.4287	12.571
161.6	7.2103	12.79
161.6	7.144	12.856
161.6	7.1429	12.857

Table 5.1: Table showing the solution for 2-edged network with fixed budgeted capacity $c = 20$ and 4 iterations.

z-value	x_1	x_2
181.6	3.3333	6.6667
181.34	3.815	6.185
181.2	3.5622	6.4378
181.2	3.5714	6.4286
181.2	3.5714	6.4286

Table 5.2: Table showing the solution for 2-edged network with fixed budgeted capacity $c = 10$. We reduce the budgeted capacity to let the initial z indicate a congestion scenario. The optimal solution however indicate no congestion. The algorithm takes 4 iterations to complete.

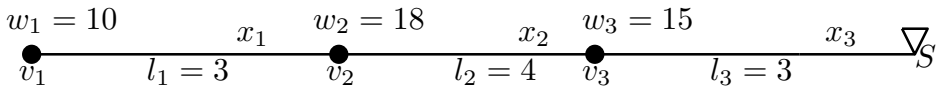

Figure 5.11: Path network with three edges. Fixed capacity at $c = 20$.

z-value	x_1	x_2	x_3
321.55	3.3333	6.6667	10
317.23	4.7522	8.3469	6.9009
317.53	4.8909	7.5867	7.5224
317.52	4.6281	7.6863	7.6856
317.52	4.6365	7.6818	7.6818
317.52	4.6365	7.6818	7.6818

Table 5.3: Table showing the solution for 3-edge network with 5 iterations.

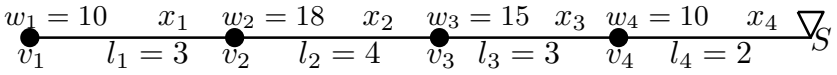

Figure 5.12: Path network with four edges.

z-value	x_1	x_2	x_3	x_4
472	2	4	6	8
448.36	3.4176	5.8985	6.4379	4.246
448.32	3.9184	6.4114	5.0478	4.6224
449.37	3.7532	5.7909	5.2318	5.2241
450.03	3.5245	5.6524	5.4125	5.4106
450.44	3.4614	5.5617	5.4885	5.4884
450.63	3.4264	5.5256	5.524	5.524
450.63	3.4255	5.5248	5.5248	5.5248
450.63	3.4255	5.5248	5.5248	5.5248

Table 5.4: Table showing the solution for 4-edged network with $c = 20$. The algorithm takes 8 iterations to complete.

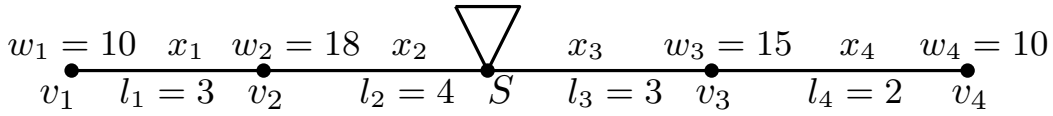

Figure 5.13: Path network with four edges and a sink in the middle.

z-value	x_1	x_2	x_3	x_4
308.68	3.3333	6.6667	6.6667	3.3333
308.31	3.7691	6.1067	6.4399	3.6843
308.13	3.6199	6.494	6.3946	3.4915
308.13	3.6163	6.5093	6.389	3.4853
308.13	3.6163	6.5094	6.389	3.4853

Table 5.5: Solution for the case where the sink is in the middle of the path. The solutions are monotone from either sides. The algorithm takes 4 iterations to complete.

Chapter 6

Conclusion and future work

In this thesis we have developed and analysed novel algorithms and datastructure suitable for the evacuation planning process of sparse or remote communities. We first of all focus on the algorithm that allocates capacities optimally with the objective of minimizing the minmax (or completion time) objective function. When the topology is a path network, we are able to reduce the problem to finding an optimal allocation to the furthest edge network, and with an efficient binary search query through our constructed parametric equation and segment tree datastructure to compute the optimal capacities for other edges efficiently.

We extend the same idea to the case when the topology is a star network and the sink is located at the centre of the network. However, when the sink is located at a leaf of the star network, different approach proposed in [10] called the Klinz theorem is employed, only for a star network with three edges. For an extended star network with a sink on the leaf, the problem becomes too complex and require further investigation of hidden combinatorial properties.

In Chapter 5, we proceed to developing algorithm that allocates capacities optimally to the edges of path networks with the objective of minimizing the total evacuation time of all evacuees on the networks. We notice that the problem is more difficult than the minmax problem. We discover that the associated objective function is convex and monotone allocation in the optimal solution vector applies. We also discover that the objective function may change configuration from a congestion scenario to a non-congestion scenario during the optimization process. We are able to solve the problem by incorporating the idea of

sequential quadratic programming (SQP), together with monotone intial solution vector. The SQP allows us to incorporate the associated constraints and also helps in tracking the changing configuration of the objective function.

The extension of the sink location algorithms studied in this thesis can be taken as foundation for studying the problems on complex topologies such as trees. We are excited to have published the developed algorithms for capacity provisioning problems for path networks (minmax criterion) in the networks journals. We also added the idea discussed for star networks with a sink in the middle in the same publication. We were able to submit our results for the minsum problem to the EAMMO conference at the time of writing this thesis. We plan to have the results published in a reputable journal.

One could pose a question to know how the capacity provisioning problems could help emergency responders with logistics for an effective evacuation of a remote region. We consider the following steps:

(i) identify the location of sites of interest and the number of evacuees at each site

(ii) compute a steiner tree spanning the sites. This tree becomes our network. Any existing roads can also be added to the network.

(iii) decide and fix location of evacuation sink or sinks based on various logistical needs

(iv) solve a capacity provisioning problem over the designed network.

The computed capacities can then inform emergency responders in making strategic decisions concerning the allocation of emergency vehicles or aircraft. We envision that solving the problem on path and star networks seem relevant for practice, and we believe that the extension to tree topologies will offer more in reality.

We propose to embark on the following work in future:

(i) Multiple sinks capacity provisioning problems: We have an idea that when there are multiple sinks on path networks, then there exist some optimal split edges which need to be discovered for efficient algorithms.

(ii) Minmax capacity provisioning on extended star network with the sink on one of its leaf nodes: The application of the Klinz theorem gives us some ideas as we have noticed for the case of three-edged star network. However, the influence of multiple flow functions encountered in the extended star topology need a further investigation

(iii) Capacity provisioning on tree network in minmax criterion: We hope to extend our study of the minmax capacity provisioning to tree topologies. We understand that the traditional minsum sink location problem is limited to path network in literature, while the minmax problems have been extended to general graph through approximation algorithm techniques in [35]. We hope our ideas for star networks would show some light to hidden properties in tree networks.

Bibliography

[1] Niels Henrik Abel. "Démonstration de l'impossibilité de la résolution algébrique des équations générales qui passent le quatrieme degré". In: *Journal für die reine und angewandte Mathematik* 1 (1826), pp. 65–96.

[2] Jasbir Arora. *Introduction to optimum design*. Elsevier, 2004.

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

[4] Robert Benkoczi and Rajib Das. "The minmax sink location problem on dynamic cycle networks". unpublished manuscript. August 2018.

[5] 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*. Springer. 2017, pp. 133–144.

[6] Binay Bhattacharya, Yuya Higashikawa, Tsunehiko Kameda, and Naoki Katoh. "An O ($n^2 \log^2 n$) Time Algorithm for Minmax Regret Minsum Sink on Path Networks". In: *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[7] Binay Bhattacharya and Tsunehiko Kameda. "Improved algorithms for computing minmax regret sinks on dynamic path and tree networks". In: *Theoretical Computer Science* 607 (2015), pp. 411–425.

[8] Paul T Boggs and Jon W Tolle. “Sequential quadratic programming”. In: *Acta numerica* 4 (1995), pp. 1–51.

[9] Charles George Broyden, John E Dennis Jr, and Jorge J Moré. “On the local and superlinear convergence of quasi-Newton methods”. In: *IMA Journal of Applied Mathematics* 12.3 (1973), pp. 223–245.

[10] Hoppe Bruce and Tardos Éva. “The quickest transshipment problem”. In: *Mathematics of Operations Research* 25.1 (2000), pp. 36–62.

[11] Richard H Byrd and Jorge Nocedal. “An analysis of reduced Hessian methods for constrained optimization”. In: *Mathematical Programming* 49.1 (1990), pp. 285–323.

[12] Di Chen and Mordecai Golin. “Sink evacuation on trees with dynamic confluent flows”. In: *27th International Symposium on Algorithms and Computation (ISAAC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[13] Di Chen and Mordecai J Golin. “Minmax Centered k-Partitioning of Trees and Applications to Sink Evacuation with Dynamic Confluent Flows”. In: *arXiv preprint arXiv:1803.09289* (2018).

[14] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. Chapter 8.2. MIT press, 2009.

[15] Daniel Dressler and Martin Strehler. “Capacitated confluent flows: complexity and algorithms”. In: *International Conference on Algorithms and Complexity*. Springer. 2010, pp. 347–358.

[16] Roger Fletcher. “The sequential quadratic programming method”. In: *Nonlinear optimization*. Springer, 2010, pp. 165–214.

[17] J Flum and M Grohe. *Parameterized Complexity Theory. 2006. Texts in Theoretical Computer Science. An EATCS Series, 2006*.

[18] Greg N Frederickson and Donald B Johnson. “Finding kth paths and p-centers by generating and searching good data structures”. In: *Journal of Algorithms* 4.1 (1983), pp. 61–80.

[19] Philip E Gill and Elizabeth Wong. “Sequential quadratic programming methods”. In: *Mixed integer nonlinear programming*. Springer, 2012, pp. 147–224.

[20] Mordecai J Golin, Hadi Khodabande, and Bo Qin. “Non-approximability and polylogarithmic approximations of the single-sink unsplittable and confluent dynamic flow problems”. In: *arXiv preprint arXiv:1709.10307* (2017).

[21] Shih-Ping Han. “Superlinearly convergent variable metric algorithms for general nonlinear programming problems”. In: *Mathematical Programming* 11.1 (1976), pp. 263–282.

[22] Shih-Ping Han. “A globally convergent method for nonlinear programming”. In: *Journal of optimization theory and applications* 22.3 (1977), pp. 297–309.

[23] Yuya Higashikawa, Mordecai J Golin, and Naoki Katoh. “Minimax regret sink location problem in dynamic tree networks with uniform capacity”. In: *International Workshop on Algorithms and Computation*. Springer. 2014, pp. 125–137.

[24] Yuya Higashikawa, Mordecai J Golin, and Naoki Katoh. “Multiple sink location problems in dynamic path networks”. In: *Theoretical Computer Science* 607 (2015), pp. 2–15.

[25] Yuya Higashikawa and Naoki Katoh. “A Survey on Facility Location Problems in Dynamic Flow Networks”. In: *The Review of Socionetwork Strategies* 13.2 (2019), pp. 163–208.

[26] Bruce Hoppe and Éva Tardos. “Polynomial Time Algorithms for Some Evacuation Problems.” In: *SODA*. Vol. 94. 1994, pp. 433–441.

[27] Naoyuki Kamiyama, Naoki Katoh, and Atsushi Takizawa. “Theoretical and practical issues of evacuation planning in urban areas”. In: *The Eighth Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA2007)*. 2007, pp. 49–50.

[28] Van Marc Kreveld, Otfried Schwarzkopf, Mark de Berg, and Mark Overmars. *Computational geometry algorithms and applications*. Springer, 2000.

[29] Ford Jr R. Lester and Fulkerson Delbert Ray. “Constructing maximal dynamic flows from static flows”. In: *Operations research* 6.3 (1958), pp. 419–433.

[30] Satoko Mamada, Takeaki Uno, Kazuhisa Makino, and Satoru Fujishige. “An O ($n \log^2 n$) algorithm for a sink location problem in dynamic tree networks”. In: *Exploring New Frontiers of Theoretical Informatics*. Springer, 2004, pp. 251–264.

[31] Nimrod Megiddo. “Combinatorial optimization with rational objective functions”. In: *Math and Operation Research* 4 (1979), pp. 414–424.

[32] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Vol. 31. OUP Oxford, 2006.

[33] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[34] UM Palomares and Olvi L Mangasarian. “Superlinearly convergent quasi-Newton algorithms for nonlinearly constrained optimization problems”. In: *Mathematical Programming* 11.1 (1976), pp. 1–13.

[35] Belmonte Rémy, Higashikawa Yuya, Katoh Naoki, and Okamoto Yoshio. “Polynomial-time approximability of the k-Sink Location problem”. In: *arXiv preprint arXiv:1503.02835* (2015).

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

[37] Benkoczi Robert, Bhattacharya Binay, Higashikawa Yuya, Kameda Tsunehiko, and Katoh Naoki. “Minsum k-sink problem on path networks”. In: *Theoretical Computer Science* (2019).

[38] Mamada Satoko, Uno Takeaki, Makino Kazuhisa, and Fujishige Satoru. “An $O(n \log^2 n)$ algorithm for the optimal sink location problem in dynamic tree networks”. In: *Discrete Applied Mathematics* 154.16 (2006), pp. 2387–2401.

[39] F Bruce Shepherd and Adrian Vetta. “The inapproximability of maximum single-sink unsplittable, priority and confluent flow problems”. In: *arXiv preprint arXiv:1504.00627* (2015).

[40] Richard A Tapia. “Diagonalized multiplier methods and quasi-Newton methods for constrained optimization”. In: *Journal of Optimization Theory and Applications* 22.2 (1977), pp. 135–194.

[41] Robert B Wilson. “A simplicial algorithm for concave programming”. In: *Ph. D. Dissertation, Graduate School of Bussiness Administration* (1963).