Universitatea
Transilvania
din Brașov

Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

INTERDISCIPLINARY DOCTORAL SCHOOL

Faculty of Matemathematics and Computer Science

Luciana CĂRĂBĂNEANU
(căsătorită MAJERCSIK)

INCREMENTAL METHODS IN DYNAMIC NETWORKS

SUMMARY

Scientific supervisor

Prof. dr. Marius-Sabin TĂBÎRCĂ

Brașov, 2024

Acknowledgements

I would like to express my deepest gratitude to everyone who has supported me throughout the research and completion of this thesis.

First and foremost, I am deeply grateful to my supervisor, Dr. Sabin Tabirca, for his invaluable guidance, encouragement and support. His insightful knowledge, feedback and ideas have been fundamental in shaping this thesis, and his patience and understanding have been a source of motivation throughout my research journey.

I would also like to express my sincere gratitude to the members of my thesis committee, Dr. Laura Ciupala, Dr. Eleonor Ciurea and Dr. Dorin Bocu, for their constructive criticism and suggestions, which have greatly improved the quality of this thesis.

I am also grateful to my colleagues and friends in the department and research group, especially Dr. Adrian Deaconu, who provided a stimulating and supportive environment. The discussions, collaborations and shared experiences have been an integral part of my academic journey.

Finally, I dedicate this thesis to my family and friends, whose unwavering belief in my abilities provided the strength and motivation needed to complete this work. Their constant support, love and encouragement have been invaluable and I am eternally grateful for their presence in my life.

I am afraid I lack the words to express my gratitude to those who had faith in me and encouraged me. Your unwavering support, patience and love means more than I can express.

Thank you all for everything.

"Every act of conscious learning requires the willingness to suffer an injury to one's self-esteem. That is why young children, before they are aware of their own self-importance, learn so easily." — Thomas Szasz

Chapter 1

Introduction

Many industries have been transformed by the rapid growth of connected data in recent decades. Many of the biggest companies, including Amazon, Google, Meta, Instagram, LinkedIn, Pinterest and Airbnb, have put data and how it relates to them at the heart of how they operate. This data, which is often represented as a graph or a network, is vast and dynamic, and requires efficient management strategies.

Dynamic networks are subject to changes occurring in a variety of ways, including modifications to individual arcs or nodes or more extensive changes that affect several parts of the network simultaneously. For instance, an individual arc's weight may be increased or decreased, a node may be added or removed, or several connections may be altered due to circumstances. Efficiently managing these changes is important for enabling the network to adapt and maintain optimal performance.

To meet the challenges posed by dynamic data, it is essential to have algorithms that can efficiently handle changes in network structure. These algorithms are designed to update only those parts of the network that are affected, thus avoiding the computational burden of having to re-calculate solutions from the start. By focusing on incremental changes, these algorithms significantly improve the ability to effectively manage dynamic networks. This ensures that systems remain both performant and adaptive. Such algorithms are essential in a wide range of applications, including the optimization of flows through the network, the maintenance of shortest paths, the dynamic adjustment of configurations of nodes through clustering algorithms, and the identification of unusual patterns in real time through anomaly detection algorithms. They allow for rapid adaptation to change, ensuring that systems remain efficient and responsive to changes in the environment.

1.1 Motivation

In today's interconnected world, large-scale networks are everywhere, spanning social media platforms, transportation systems, telecommunications infrastructures, and more. Their size and dynamic nature require scalable algorithms that can efficiently manage and optimize their functioning. As these networks continue to grow and evolve, the ability to quickly adapt to changes is becoming more and more important in order to maintain optimal performance.

Two types of algorithms often appear intertwined in various aspects of dynamic network management, namely shortest path algorithms and maximum flow algorithms. The relationship between them is remarkable, although they focus on different optimization objectives. Maximum flow algorithms optimize overall network utilization by ensuring the maximum possible flow from a source to a sink, while shortest path algorithms aim to minimize the total travel cost or distance between nodes. Given the wide applicability and interconnected nature of these problems, developing efficient algorithms that can handle dynamically changing network conditions is essential.

Incremental changes in the network, such as adjustments to arc weights or node additions and deletions have a considerable effect on the computations of shortest paths and maximum flows. Addressing these changes incrementally, rather than recalculating solutions from the beginning is essential for maintaining an optimal performance. The choice to focus on maximum flow and single source shortest path (SSSP) algorithms is motivated by their fundamental importance in various real-world applications and their ability to address critical challenges in network optimisation and management. The interaction between these algorithms highlights their complementary roles. By exploring and advancing these algorithms, we can improve the ability of large-scale dynamic networks to operate efficiently and to withstand or quickly recover from changes or disruptions. This involves ensuring that networks can handle fluctuations in demand, adapt to new conditions and continue to operate smoothly without significant degradation in performance.

1.2 Objectives

The main objectives of this thesis are:

- **Investigate existing algorithms**. The first objective is to thoroughly investigate the current state-of-the-art algorithms for incremental flow and dynamic single source shortest path (SSSP) problems. This will include an in-depth analysis of their theoretical foundations, computational complexity and the strategies they employ to deal with dynamic network changes. Identifying the strengths and weaknesses of these existing algorithms will highlight their limitations and potential improvements. In addition, understanding their practical applications will provide insight into how these algorithms perform in different real-world scenarios and highlight specific use cases where they are particularly effective.

- **Develop efficient dynamic algorithms**. The next objective is to propose new algorithms that can efficiently manage dynamic changes in network conditions or demands, based on the insights gained from the study of existing algorithms. These new methods will focus on minimizing the computational complexity by updating only the affected parts of the network, thus ensuring optimal performance. The goal is to develop algorithms that are not only theoretically sound, but also practical. They must be able to handle large networks while maintaining high performance and scalability.

- **Application to real-world problems**. Another objective of this thesis is to examine the potential for integrating remote sensing data with dynamic networks in a variety of scenarios designed to study phenomena that over time produce visible changes on the Earth's surface. This objective underscores the relevance and applicability of the research to real-world problems and highlights the practical benefits and potential impact of the proposed solutions.

1.3 Outline of the Thesis

The present thesis is structured into the following chapters:

Chapter 2: Theoretical Background

This chapter provides the basic concepts and theoretical background necessary to understand the dynamic and incremental algorithms related to the shortest path problem and the maximum flow problem. It covers the basics of network flow problems, shortest path algorithms, and relevant computational techniques.

Chapter 3: Scoping Literature Review

This chapter reviews current state-of-the-art algorithms for dynamic SSSP and incremental flow problems. It includes a detailed discussion of the most well-known solutions, the methodologies they use, and the applications they are used in.

Chapter 4: Dynamic Adjustment of Single Source Shortest Paths

This chapter introduces a new dynamic SSSP algorithm. It describes how it handles changes in edge weights and efficiently maintains the shortest path tree. It uses a methodology that minimises the computational complexity and ensures that updates are performed quickly. The correctness and complexity of the algorithm are thoroughly investigated, providing a detailed analysis of its theoretical foundations. In addition, several examples are presented to illustrate the practical application and performance of the algorithm, showing how it adapts to changes in the network while maintaining accurate shortest path calculations.

Chapter 5: Maximum Flow through Network Expansion

This chapter examines the practical and theoretical aspects of the maximum flow problem with a focus on network expansion. Often network expansion is necessary to increase the amount of flow transported through a network. Starting from a given network G, the objective of the Minimum Cost Network Expansion Problem (MCNEP) studied in this chapter is to obtain a new network G by increasing the arc capacities within given limits or by adding new arcs. The goal is to minimise the cost of the changes while allowing G' to transport w units of flow from the source to the sink.

We present a new algorithm designed to solve the MCNEP, provide detailed proofs of its correctness, and estimate its computational complexity. Several examples illustrate the application of the algorithm, demonstrating its efficiency in optimizing network expansions to meet increased flow demands while minimizing costs. This chapter highlights the practical utility of the proposed algorithm in addressing the problem of dynamic flow augmentation in various networked systems.

Chapter 6: Integration of Remote Sensing Data with Dynamic Network Processing

The objective of this chapter is to introduce a method for integrating remote sensing data with dynamic networks and using them in combination in a dynamic vegetation monitoring algorithm. The proposed approach can be used in agriculture or environmental monitoring, potentially in combination with in situ measurements or devices, in order to enable timely intervention in unfavorable situations, due to natural phenomena or human interventions. The applications discussed in this chapter were partially supported by the AI4AGRI project, entitled “Ro-

manian Excellence Center on Artificial Intelligence on Earth Observation Data for Agriculture," which received funding from the European Union's Horizon Europe research and innovation program under grant agreement no. 101079136.

Chapter 7: Conclusions

The final chapter summarizes the findings of the thesis and reflects on the progress made. It discusses potential future research directions, emphasizing the importance of developing robust algorithms that perform efficiently in a variety of applications and dynamic scenarios, and are scalable for use in large networks.

Chapter 2

Theoretical Background

2.1 Introduction

The field of network optimization represents an important area within operations research and computer science, focusing on the analysis and optimization of the flow of resources, data, or entities through networks represented as graphs. This field of study addresses a number of important problems, including determining the maximum amount of flow that can be transmitted through a network (maximum flow problem) and the identifying the shortest paths between nodes (shortest path problem). The significance of these problems lies in their extensive practical applications in various real-life situations.

The relationship between maximum flow and shortest path algorithms is deep, despite their focus on distinct optimization objectives. Maximum flow algorithms prioritize the optimization of overall network capacity utilization, whereas shortest path algorithms seek to identify the most efficient route or path between two nodes, with the objective of minimizing total cost or distance traveled. Computational similarities exist between the two in the sense that shortest path algorithms can be related to maximum flow algorithms. This includes the methods of finding augmenting path, which are used in Ford-Fulkerson and Edmonds-Karp algorithms. Furthermore, in practical application, solutions to these problems often complement each other. The following examples illustrate this.

In this next section, our focus will turn to the specific definitions, terminology, and types of problems associated with the shortest path problem.

2.2 The Shortest Path Problem

The definitions and terminologies presented here are based on [1].

Definition 1. *A directed graph $G = (V, A)$ consists of a set V of vertices (nodes) and a set $A = \{(i, j) \mid i \in V, j \in V\}$ of ordered pairs of distinct nodes. A directed network or directed weighted graph is a directed graph where each arc has an associated numerical value (cost or capacity) determined by the function*

$$w : A \to \mathbb{R} \tag{2.1}$$

known as the weight function. In this case, the graph is denoted $G = (V, A, w)$.

Definition 2. *A directed arc (i, j) has two endpoints i and j: i is referred to as the start node of the arc and j is referred to as the end node of the arc. We say that the arc (i, j) is incident to nodes i and j. Additionally, the arc (i, j) is an outgoing arc of node i and an incoming arc of node j. If an arc $(i, j) \in A$, we say that the node j is adjacent to node i.*

Definition 3. *A graph $G' = (N', A')$ is a subgraph of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. A graph $G' = (N', A')$ is a spanning subgraph of $G = (N, A)$ if $N' = N$ and $A' \subseteq A$.*

Definition 4. *A walk in a directed graph is a sequence of $n_1 - a_1 - n_2 - a_2 - ... - a_{k-1} - n_k$ with $k \geq 2$ such that the i-th arc in the sequence is either $a_i = (n_i, n_{i+1})$, in which case is called a forward arc, or $a_i = (n_{i+1}, n_i)$, in which case is called a backward arc.*

A *directed walk* is a walk where for any two consecutive nodes n_i and n_{i+1} of the walk, $(n_i, n_{i+1}) \in A$.

Definition 5. *A directed path is a directed walk without any repetition of nodes.*

We can store a directed path by defining a *predecessor index* $p(j)$ for any node in the path. If i and j are two consecutive nodes on the path, then $p(j) = i$.

In a weighted directed graph $G = (V, A, w)$ we define the *length of a directed path* as the sum of lengths of the arcs in the path.

Definition 6. *A directed cycle is a directed path for which the start node and the and node are the same.*

Definition 7. *A graph is acyclic it it contains no directed cycles.*

Definition 8. *A directed out-tree rooted at node s is a tree such that the unique path in the tree from s to any other node is a directed path.*

The Shortest Path Problem

In a weighted directed graph $G = (V, A, w)$, the shortest path problem is concerned with identifying the most efficient path between nodes based on arc weights. In particular, the shortest path problem seeks to determine a path where the sum of arc weights (or costs) is minimized, given a source node s and a target node t. In addressing the shortest path problem, the following assumptions are made:

1. The graph is directed.
2. The graph does not contain any negative cycles, meaning that there are no directed cycles with negative total weights.
3. There exists a directed path from a specified source node, designated as s, to every other node in the graph.

Over the years, researchers have explored a number of distinct types of shortest path problems in graph theory, including:

- **The single source shortest path (SSSP) problem**, which seeks to identify the shortest paths from a single source node to all other nodes in the network.
- **The single pair shortest path (SPSP) problem**, where the objective is to determine the shortest path between two specific nodes in the network.
- **The all pairs shortest path (APSP) problem** which is concerned with the computation of the shortest path between every pair of nodes in the graph.

There are also a number of specialized versions of the shortest path problem: the maximum capacity path problem, the maximum reliability path problem, the shortest paths with additional constraints or the resource constrained shortest path problem. Each of these variations addresses a specific challenge or application.

2.3 The Maximum Flow Problem

This section will introduce the fundamental concepts of maximum flow, minimum cut, and minimum cost flow, which will be crucial for understanding the techniques and algorithms discussed in subsequent chapters. It introduces also the key concepts needed to understand how network flow algorithms work. The definitions and theorems presented here are based on [1].

Let $G = (V, A, s, t, u)$ be an $s - t$ directed network, where V is the set of $n > 0$ vertices (nodes), and $A \subseteq V \times V$ is the set of $m \geq 0$ arcs (directed edges). Each arc $a = (i, j) \in A$ connects two nodes i and j from V, s is a special node called source, and t is a node called sink. In G, we define the capacity function $u : A \to \mathbb{R}_+^*$. The value $u(a)$ is the maximum flow that can be transported from node i to node j on the arc $a = (i, j) \in A$.

Definition 9. *A (feasible) flow in an $s-t$ directed network G is a function $f : A \to \mathbb{R}_+$ satisfying the boundary restrictions (2.2) and the conservation conditions from (2.3).*

$$0 \leq f(a) \leq u(a), \forall a = (i, j) \in A \tag{2.2}$$

$$\sum_{\substack{j \in V, \\ (i,j) \in A}} f(i, j) - \sum_{\substack{j \in V, \\ (j,i) \in A}} f(j, i) = \begin{cases} 0, \textit{if } i \in V - \{s, t\} \\ v_f, \textit{if } i = s \\ -v_f, \textit{if } i = t \end{cases} \tag{2.3}$$

where $v_f = v(f) \geq 0$ is called the value of the flow f.

Definition 10. *A feasible flow f^* is a maximum flow if it has the maximum value among all the feasible flows in the network G, i.e.:*

$$v(f^*) = max\{v(f) \mid f \textit{ is a feasible flow in } G\} \tag{2.4}$$

In (2.4), $v(f^*)$ represents the maximum amount of flow that can be transported in the network G, from s to t.

The concept of the residual network played a key role in the development of all the maximum flow algorithms we will be discussing. Therefore, we shall continue by defining the resi - dual network as follows.

Definition 11. *Given a feasible flow f, the residual capacity of any arc $(i, j) \in A$ represents the maximum additional flow that can be sent from node i to node j using the arcs (i, j) and (j, i). The residual capacity of arc (i, j) has two components:*

1. $u(i,j) - f(i,j)$ *which represents the unused capacity of arc* $a = (i,j) \in A$*;*

2. $f(j,i)$ *which represents the flow on arc* (j,i) *that can be reduced to increase the flow from node* i *to node* j*.*

Therefore, the residual capacity is calculated as

$$r(i,j) = u(i,j) - f(i,j) + f(j,i). \tag{2.5}$$

and the residual network with respect to the flow f, $\widetilde{G}(f) = (V, \widetilde{A}, s, t, r)$ *consists only of arcs with positive residual capacities, i.e.* $\widetilde{A} = \{(i,j) \in A \mid r(i,j) > 0\}$*.*

Definition 12. *In a residual network* $\widetilde{G}(f) = (V, \widetilde{A}, s, t, r)$*, a function* $d : V \to \mathbb{Z}_+$ *is called a distance function. The distance function d is considered valid if it satisfies the following two conditions:*

$$d(t) = 0 \tag{2.6}$$

$$d(i) \leq d(j) + 1, \textit{ for any } (i,j) \in \widetilde{A}. \tag{2.7}$$

The value $d(i)$ is called the *distance label* of node i, and the conditions 2.6 and 2.7 are called *validity conditions*.

It is known that if the distance labels are valid, then the distance label $d(i)$ represents a lower bound for the shortest path length from node i to node t in the residual network $\widetilde{G}(f)$.

Definition 13. *Distance labels are said to be exact if, for each node* i*,* $d(i)$ *is equal to the length of the shortest path from node* i *to node* t *in the residual network* $\widetilde{G}(f)$*.*

Definition 14. *An arc* (i,j) *in the residual network* $\widetilde{G}(f)$ *is called admissible if it satisfies the condition*

$$d(i) = d(j) + 1 \tag{2.8}$$

otherwise the arc (i,j) *is called inadmissible.*

A path from the source node to the sink node in the residual network $\widetilde{G}(f)$ is called *admissible* if it contains only admissible arcs; otherwise it is called *inadmissible*.

Definition 15. *A preflow in an* $s-t$ *directed network* G *is a function* $f : A \to \mathbb{R}_+$ *satisfying the boundary restrictions (2.2) and a relaxation of the conservation conditions from (2.3).*

$$e(i) = \sum_{\substack{j \in V, \\ (j,i) \in A}} f(j,i) - \sum_{\substack{j \in V, \\ (i,j) \in A}} f(i,j) \geq 0, \forall i \in V - \{s,t\} \tag{2.9}$$

For a given preflow function f, the difference between the inflow and the outflow, $e(i)$ is called *the excess* of the node $i \in V - \{s, t\}$. If a node has positive excess then it is called an *active* node.

Definition 16. *A pseudoflow in an $s-t$ directed network G is a function $f : A \to \mathbb{R}_+$ satisfying only the boundary restrictions (2.2).*

For any pseudoflow, f we define the *imbalance* of node i as:

$$e(i) = \sum_{\substack{j \in V, \\ (j,i) \in A}} f(j,i) - \sum_{\substack{j \in V, \\ (i,j) \in A}} f(i,j), \forall i \in V \tag{2.10}$$

We say that i is an *excess node* if $e(i) > 0$ and a *deficit node* if $e(i) < 0$. If $e(i) = 0$, node i is *balanced*. Consequently, a preflow is a particular case of pseudoflow.

To further understand the network flow concepts, we introduce the concept of an s–t cut and the min-cost flow problem.

Definition 17. *An s–t cut in a network $G = (V, A, s, t, u)$ is a partition of V into two disjoint subsets S and T such that $s \in S$ and $t \in T$. The capacity of the s–t cut, denoted as $c(S, T)$, is the sum of the capacities of the arcs from S to T:*

$$c(S,T) = \sum_{(i,j) \in S \times T} u(i,j). \tag{2.11}$$

The Max-Flow Min-Cut Theorem states that the value of the maximum flow in a network is equal to the capacity of the minimum s–t cut in the network. This theorem establishes a fundamental relationship between flows and cuts in network theory.

Theorem 1 (Max-Flow Min-Cut Theorem [1])**.** *In any flow network, the maximum value of the flow is equal to the capacity of the minimum cut that separates the source and the sink.*

While the max-flow problem concerns the identification of the maximum feasible flow within a network, the min-cost flow problem is directed towards establishing the least costly means for the transmission of a given volume of flow through a network. The min-cost flow problem can be regarded as an extension of the max-flow problem, where each arc is associated with a cost factor reflecting the cost of sending flow through it and it is defined as follows:

Definition 18. *The Min-Cost Flow Problem in a directed network G=(V,A) with a capacity function $u : A \to \mathbb{R}_+^*$ and a cost function $c : A \to \mathbb{R}_+$ seeks to find a flow $u : A \to \mathbb{R}_+$ that*

minimizes the total cost:

$$\min \sum_{(i,j)\in A} c(i,j)f(i,j) \tag{2.12}$$

subject to:

$$0 \leq f(i,j) \leq u(i,j), \forall (i,j) \in A \tag{2.13}$$

and where:

$$\sum_{j\in V,(i,j)\in A} f(i,j) - \sum_{j\in V,(j,i)\in A} f(j,i) = b(i), \forall i \in V \tag{2.14}$$

with b(i) being the supply/demand at node i. If b(i)>0, node i is a supply node, and if b(i)<0, node i is a demand node.

In general, the following assumptions are made:

1. All data are integers.
2. The network is directed.
3. The sum of all excesses (supplies) and deficits (demands) is zero:

$$\sum_{i\in V} b(i) = 0 \tag{2.15}$$

4. All arc cost are nonnegative.

To conclude, this chapter has presented the principal definitions and terminologies associated with the shortest path problem and maximum flow problem. This has established a foundation for understanding the computational methodologies employed to address these issues.

Efficient static algorithms for maximum flow and shortest path problems have been a focus of research for years. The development of incremental and dynamic network algorithms arises from the recognition of the limitations of static algorithms in addressing real-life scenarios where network characteristics or flow requirements change over time. Although static algorithms provide valuable computational techniques for maximum flow and shortest path problems, they operate under the assumption of fixed network structures and capacities. However, in dynamic environments such as transportation systems, communication networks, or supply chains, these assumptions frequently do not hold true. Therefore, incremental and dynamic algorithms are necessary to accommodate the dynamic nature of such systems. Incremental and dynamic algorithms are two types of algorithms used to update the solutions

in response to changes in the network. Incremental algorithms efficiently update the solutions in response to small changes in the network, such as edge additions or capacity/weight adjustments. Dynamic algorithms, on the other hand, can adapt to variations such as edge insertions, deletions, or capacity adjustments, ensuring that the solutions remain effective as the network evolves. Dynamic algorithms aim to maintain optimal or near-optimal solutions despite changes in network topology. By incorporating temporal dynamics into the optimization problem, incremental and dynamic algorithms offer flexibility and adaptability, enabling them to effectively respond to changing network conditions. This adaptability is crucial for applications such as real-time traffic management and dynamic resource allocation in evolving environments. Therefore, it is essential to use incremental and dynamic algorithms to address the complexities of dynamic network scenarios. Chapter 3 of the thesis will discuss the state of the art incremental (dynamic) algorithms.

Chapter 3

Scoping Literature Review

3.1 Introduction

Static algorithms for single source shortest path problems (SSSP) typically assume that the network remains constant over time. However, this standard approach is not suitable for real-life situations where conditions are likely to change as a result of an evolving environment. In such cases, dynamic SSSP algorithms are specifically designed to handle changes, such as updates to edge weights, by incrementally adjusting the shortest path tree (SPT) rather than recalculating it from scratch. This method is particularly important in scenarios where the network experiences incremental changes and requires adaptive strategies to maintain optimal paths without requiring a complete recalculation.

Dynamic SSSP algorithms efficiently update the shortest path tree in response to changes in edge weights. These algorithms incrementally adjust only the affected parts of the network, significantly improving time complexity and enabling rapid adaptation to changing conditions. This selective updating allows for rapid responses to changing network conditions, making dynamic SSSP algorithms crucial for applications where network dynamics are frequent.

In our review, we examine the fundamentals of dynamic SSSP algorithms, their theoretical basis, and their importance in practical applications where network changes are a frequent phenomenon. We explore how these algorithms minimize computational complexity by selectively updating only those parts of the network directly affected by changes, thus ensuring efficient recalculations of shortest path trees.

3.2 Dynamic Single Source Shortest Path Algorithms

Although there is a considerable amount of literature on shortest path problems, in this chapter of the thesis, we will focus on the most well-known solutions for the single source shortest path problem in dynamic contexts. The dynamic maintenance of single source shortest paths, or equivalently of the shortest path tree (SPT), involves maintaining an optimal shortest path tree in a graph as its edge weights change over time. This problem is important in a series of applications, such as network routing and urban traffic management. A set of algorithms have been proposed to address this problem in an efficient manner, utilizing diverse methodologies to accommodate dynamic updates. A number of these will be discussed in this section.

Ramalingam and Reps node consistency based algorithm

In 1996, Ramalingam and Reps [19] introduced an iterative algorithm, which is referred to as Dynamic SWSF-FP. This algorithm was designed to handle edge insertions and deletions in a dynamic graph, but it can be adapted to manage also weight increases and decreases. The fundamental concepts of their approach involve defining nodes and edges in terms of **consistency**. The main idea of this algorithm is to maintain a consistent shortest-path tree by using priority queue Q to manage updates efficiently. A node j is termed as consistent if its distance $d[j]$ matches the minimum distance obtained by traversing any incoming edge (i, j). Formally, this consistent value $con(j)$ is given by:

$$\text{con}(j) = \begin{cases} \min\limits_{(i,j)\in A} \{d[i] + w(i,j)\}, & \text{if } j \neq s \\ 0, & \text{if } j = s \end{cases} \tag{3.1}$$

A node is defined as over-consistent if $d[j] > con(j)$. An edge (i, j) is defined as consistent if $d[j] = w(i, j) + d[i]$, and underconsistent if $d[j] > w(i, j) + d[i]$.

The algorithm maintains auxiliary data in order to track whether or not an edge lies on the shortest path subtree, SPT, which is rooted at the source node s. In addition, the number of incoming edges for each node in SPT is also recorded. The steps of the algorithm can be briefly summarized as follows:

- **Processing weight decreases**: The algorithm updates the edge length, relaxes the edge, and processes nodes to maintain consistency in the shortest-path tree. When an edge

weight decreases, the algorithm first updates the length of the edge and relaxes it. If the edge is underconsistent, it updates the distance of the node and inserts it into a priority queue Q.

- **Processing weight increases**: The algorithm updates the edge length and removes the edge from SPT if it lies within it. Upon the deletion of the edge, a subtree B of the SPT is no longer connected to s. Each edge from B is deleted from the SPT. For each node i from B that is now affected by the change, an update of the distances is performed. This is done by using only unaffected nodes adjacent to i, and inserting them into Q as necessary.

- **Main phase**: The main phase of the algorithm involves processing the nodes from Q by extracting the minimum node, adjusting distances, and updating or inserting edges in SPT until Q is empty. Then, the main phase of the algorithm proceeds as before.

Algorithm of Frigioni et al

In 2000, Frigioni and al. [8] proposed an iterative algorithm that is similar in nature as Ramalingam and Reps' algorithm, except that more complex auxiliary data are used, giving better theoretical worst case bounds. Their approach uses a k**-bounded accounting function on** G, which is a function $K : A \to V$ such that for each edge (i, j), the node $K(i, j)$ is either i or j and no more than k edges are associated with any node i.

The algorithm stores the k-bounded accounting function K of the graph. For a given node i, the set of edges (i, j) with $K(i, j) = i$ is referred to as ownership(i). The set of other edges adjacent to i is called not ownership(i). The backward level of an edge (k, l) and of vertex l relative to vertex k is defined as $b\_level_k(l) := d[l] - w(k, l)$. The forward level of an edge (k, l) and of vertex l relative to vertex k is defined as $f\_level_k(l) := d[l] + w(k, l)$.

For each node i, the algorithm maintains two priority queues, each of them containing the edges from not ownership(i). The queue B_i is max-based, with the priority of an edge (i, j) given by $b\_level_i(j)$ while The queue F_i is min-based, with the priority of an edge (i, j) given by $f\_level_i(j)$. Additionally, a shortest-path tree is maintained by storing a parent node $p[i]$ for each node $i \neq s$. The steps of the algorithm can be briefly summarized as follows:

- **Processing weight decreases**: Given an edge (i, j) with a weight decrease, the algorithm updates $w(i, j)$ and the queues B_i, F_i, B_j, and F_j. If (i, j) is underconsistent, it sets $d[j] = d[i] + w(i, j)$ and inserts j with priority $d[j]$ into Q.

- **Processing weight increases**: Given an edge (i, j) with a weight increase, the algorithm updates $w(i, j)$ and the queues B_i, F_i, B_j, and F_j. The tentative shortest-path subgraph SPT is given implicitly by all consistent edges. Upon the deletion of the edge having increased weight from the SPT, a subtree B of the SPT is no longer connected to s. Each edge from B is deleted from the SPT. For each node j from B that is now affected by the change, an update of the distances is performed $d[j] := \min\{d[i] + w(i, j) \mid (i, j) \in A, i \notin B\}$, followed by the insertion of j into Q.

- **Main phase**: The main phase proceeds as follows until Q is empty:
  - Extract and delete the minimum node j from Q.
  - Update the queues $B(j)$ and $F(j)$.
  - Check each edge (j, k) in not ownership(j) and each edge (j, k) in ownership(j) with $b\_level_j(k) > d[j]$. If (v, w) is underconsistent, set $d[k] = d[j] + w(j, k)$ and insert k with priority $d[k]$ into Q.

This algorithm efficiently handles dynamic changes by leveraging complex auxiliary data and k-bounded accounting functions to maintain the shortest path tree structure with better worst-case performance bounds.

Algorithm of Narvaez et al.

In 2000 Narvaez et al. [12] proposed a flexible batch processing algorithm, which incorporates different data structures and processing variants to optimize the handling of dynamic changes. This algorithm offers two degrees of freedom: the choice of the priority queue type (FIFO, heap, or D'Esopo-Pape) and two main phase variants. The basic idea of this algorithm is to provide flexibility and efficiency in processing dynamic updates by using different data structures and processing methods. The algorithm maintains a shortest-path tree SPT of the graph by recording the parent node $p[j]$ for each node j. The set $B(i, j)$ denotes the collection of nodes in the branch of SPT that begins with the edge (i, j), excluding i. During the algorithm's execution, a tentative parent $p'[i]$ is stored for each node i in the priority queue Q. The steps of the algorithm can be briefly summarized as follows:

- **Initialization phase**: The initialization phase handles weight increases and decreases iteratively.
  - For each edge (i, j) with weight increases Δ, the algorithm updates the length of each affected edge $w(i, j) = w(i, j) + \Delta$ and adjusts the distances of nodes in the

branch $B(i,j)$, $d[k] := d[k] + \Delta$ for each $k \in B(i,j)$, and enqueues the affected nodes. The set of all vertices with previously incremented weights is denoted as N_{inc}. Subsequently, each edge with a target in N_{inc} is relaxed, and each overconsistent node $k \in N_{\text{inc}}$ is inserted into Q with priority con(k), and $p'[k]$ is updated accordingly.

- For weight decreases, the algorithm similarly updates lengths, adjusts distances, and enqueues nodes for further updates. Specifically, it updates the length of each affected edge, sets $d[k] := d[k] - \Delta$ for each $k \in B(i,j)$, and denotes the set of all vertices with previously decremented weights as N_{dec}. Each edge with a source in N_{dec} is then relaxed, and each node k for which $d_l := \min\{d[k] + w(k,l) \mid k \in N_{\text{dec}}\} < d[l]$ is inserted into Q with priority d_l, and $p'[l]$ is updated accordingly.

- **Main phase**:

  - **First variant**: Nodes are processed from the queue by extracting the next node j, setting its distance $d[j]$ to the priority of j in Q, and updating its parent $p[j]$ to $p'[j]$. The algorithm then relaxes each outgoing edge (j,k). If (j,k) is underconsistent, it inserts k with priority $d[j] + w(j,k)$ into Q and sets $p'[k] = j$. If k is already in Q, the algorithm only updates the priority and $p'[k]$.

  - **Second variant**: After extracting the next node j, the algorithm identifies and processes descendants that need updating. It extracts the next node j from Q, where key(j) denotes the priority of j in Q. It sets $\lambda :=$ key$(j) - d[j]$ and $p[j] := p'[j]$. The algorithm then identifies the set N of all descendants of j in SPT. Subbranches starting with nodes i that already have key$(i) < d[i] + \lambda$ in Q are excluded from N. Each node in N with key$(i) \geq d[i] + \lambda$ in Q is removed from Q. The algorithm then relaxes each edge (j,k) outgoing from a node $j \in N$. If (j,k) is underconsistent, it inserts k with priority $d[j] + w(j,k)$ into Q and sets $p'[k] = j$. If k is already in Q, the algorithm only updates the priority and $p'[k]$.

This approach allows for significant flexibility and efficiency, adapting to different scenarios and optimizing the propagation of distance updates through the shortest-path tree.

The ball and string model

In 2001, Narvaez [13] proposed an intuitive ball-and-string model to explain the linear programming formulation of the dynamic SPT problem. The basic idea is to physically inter-

pret the shortest path problem using a visual and intuitive model. In this model, each node in the graph is represented by a ball, and each edge with a weight is represented by an inelastic string with a length equal to the weight of the edge. To ensure feasibility, the Euclidean distance between the balls must not exceed the length of the string. A string becomes **tight** when the distance between its nodes is equal to its length, and represents a consistent edge in the shortest path tree.

As weights change, the model adjusts by moving the balls to maintain the shortest path tree. Weight increases cause balls to drop, representing increased distances, while weight decreases cause balls to rise, representing shorter paths. This model helps to visualize the propagation of changes in the graph and the maintenance of valid shortest path trees.

In the execution of the algorithm, the root node is anchored in a fixed position while other nodes fall under gravity. As gravity pulls down these nodes, the resulting Euclidean distance of a node from the anchored root node is equal to the shortest distance. As the length of a string increases, the ball attached to the lower end will fall until one of its attached strings becomes tight. Conversely, as the length of a string decreases, the ball rises accordingly. The efficiency of this approach lies in the natural order of distance propagation, which minimizes computational effort.

The execution of the algorithm can be briefly summarised as follows:

- **Process the weight increases**: Update the length of each edge with a weight increase. If the edge is part of the existing SPT, mark its ending node and its reachable descendants as floating. For anchored nodes connected to floating nodes, compute potential distances and enqueue these floating nodes for further updates.
- **Process weight decreases**: Compute potential new distances for end nodes for edges with decreased weight. If the new distance is less than the old one, then the node is enqueued with its potential new parent and its new distance.
- **Main phase**: Extract the node with the smallest change from the queue and update its parent. Adjust the tree structure to reflect the new parent-child relationships and update distances. Mark nodes as anchored and remove them from the queue if necessary. Repeat until the queue is empty, ensuring that all affected nodes are updated efficiently.

3.3 Conclusions

From the existing algorithms in the literature, it can be concluded that the most challenging aspect of dynamic single source shortest path (SSSP) algorithms is managing the increase or decrease of edge weights. These operations require quickly updating the shortest path tree (SPT) without recomputing it from scratch. The efficiency of a dynamic SSSP algorithm is significantly influenced by its ability to handle these changes efficiently.

Techniques that selectively adjust only the affected parts of the SPT are essential for maintaining the efficiency of these algorithms. The use of competitive data structures, such as Fibonacci heaps or dynamic trees, plays a crucial role in ensuring rapid updates. These selective adjustments and efficient data structures help in minimizing the computational complexity, thereby enabling the algorithm to adapt swiftly to changes in the network.

Although significant progress has been made in the area of dynamic SSSP algorithms, there is still considerable potential for achieving faster running times in practical settings. A pertinent question is whether it is possible to develop robust algorithms that perform efficiently over a wide range of applications and for different types of networks, in both static and dynamic environments, while guaranteeing worst-case strongly polynomial-time complexity.

Chapter 4

Dynamic Adjustment of Single Source Shortest Paths

4.1 Introduction

The problem of computing shortest paths has constituted an important field of research within graph theory for a considerable period of time, and it remains a fundamental subroutine for numerous advanced tasks in various domains. Solving this problem is of great importance in fields including transportation, navigation, logistics, supply chain management, telecommunication, as well as urban planning. Additionally, shortest path computations are of great significance in social networks, where measuring the degree of separation between individuals is important in the context of social influence analysis and community detection. In biological networks, shortest path calculations assist in the analysis of protein interaction networks, enabling the identification of critical pathways in biological processes.

A traditional approach to the shortest path problem assumes a static graph, where arc weights and structures remain unchanged following the calculation of a shortest path. However, in dynamic environments, changes such as increases or decreases in arc weights necessitate the recomputation of shortest paths. This process can be computationally expensive and inefficient, particularly in the case of large graphs or frequent updates.

In this chapter, we introduce a dynamic approach to recompute single source shortest paths (SSSP) when an arc weight increases or decreases. In contrast to the conventional approach of recalculating the entire shortest path solution from scratch, dynamic algorithms up-

date the existing shortest path information based on the changes in the graph. If an arc weight increases, for instance, the algorithm merely needs to verify and update the affected paths. Conversely, if an arc weight decreases, the algorithm can efficiently disseminate this improvement throughout the network, updating the shortest paths as needed. To minimize recomputation time, the dynamic update of single-source shortest paths (SSSP) employs previously computed shortest paths. This approach is particularly useful in environments where the graph changes frequently, enabling real-time updates, while maintaining optimal paths. This algorithm is an extension of our ideas presented in [3].

4.2 Preliminaries

Sequential shortest path algorithms typically use iterative labeling methods, which maintain a temporary distance value $d[v]$ for all nodes. The value assigned to the temporary distance, may be either ∞ or the weight of a path from the source s to v. This temporary distance value serves as an upper bound on the actual distance between the source s and v. Temporary distances are updated through an operation called "arc relaxation": for an arc $(u, v) \in A$, $d[v] = min\{d[v], d[u] + w(u, v)\}$.

There are two main categories of labeling methods: label-setting and label-correcting. Algorithms that perform label setting, such as Dijkstra's, set the distance of a node as optimal in each iteration. This requires at most n iterations, where n is the number of nodes. In contrast, algorithms that are correcting labels, can relax arcs of non-optimal nodes and the number of iterations required may vary. All labels are considered temporary until the final step when they become permanent. Despite the superiority of worst-case bounds for label-setting algorithms versus label-correcting ones, label-correcting approaches often exhibit good practical performance, particularly for large graph classes with random arc weights. In order to exploit the advantageous practical performance of label-correcting approaches, [11] introduces a label-correcting algorithm, known as the Delta-stepping algorithm. This algorithm can be implemented in a highly efficient manner in both sequential and parallel settings for diverse classes of graphs. Furthermore, it is capable of achieving optimal linear time with high probability.

The Delta-stepping algorithm maintains eligible nodes with temporary distances, organized in an array of buckets. Each bucket corresponds to a specific distance range, with the size Δ of this range determined by the user. During each phase, the algorithm removes and pro-

cesses all nodes from the first non-empty bucket and relaxes all outgoing arcs having weights at most Δ. Arcs with higher weights are relaxed only afterwards. The choice of Δ is crucial to balance between excessive node reconsideration and excessive bucket traversal.

Since a modified version of the Delta-stepping algorithm is used in our Dynamic Adjustment Single Source Shortest Path algorithm (DynAdjSSSPA), a brief overview of first one is provided below. Detailed information on the Delta - stepping algorithm and its auxiliary functions can be found in [11].

The Delta - stepping algorithm processes a graph $G = (V, A, w)$ and a source node s in order to return the array $d[v]$ containing the minimum cost to reach any node v starting from s. The algorithm keeps the nodes still to be processed and that have non-optimal shortest distances in an array B of buckets, each bucket representing a distance range equal to Δ. The parameter Δ is a positive real number, which is also referred to as the "step size" or the "bucket width" and which determines the distribution of the nodes inside the buckets. Each bucket $B[i]$ contains the nodes to be processed in an iteration, and if, for a node $v \in V$, $d[v]$ represents the best known cost of reaching v from s, then v belongs to $B[i]$, where $i = \left\lfloor \frac{d[v]}{\Delta} \right\rfloor$.

The algorithm processes the buckets $B[i]$ one by one, starting from $i = 0$. At each iteration i, the algorithm focuses on the first bucket that is not empty (the current bucket) and removes all the nodes from it. For each node u removed, it relaxes all light arcs (with $w(u, v) \leq \Delta$) leaving from this node. This relaxation process may result in new nodes being added to the current bucket. These nodes will be processed in the next phase. Additionally, previously removed nodes may be reinserted if their temporary distance improves during the current phase.

The process of relaxing the light arcs is carried out in a loop in order to take into account possible re-insertions into the same bucket. Two auxiliary sets are used during this process: the request set Req, which stores pairs of nodes that can be reached by light arcs, together with their corresponding costs, and the set R (removed nodes), which collects all nodes from the bucket $B[i]$ that have been explored in order to avoid reprocessing of their light arcs in subsequent iterations of the inner loop. At the end of each iteration of the inner loop, the relaxation function is called on the pairs from the request set.

Relaxation of heavy arcs (with $w(u, v) > \Delta$) is postponed during the processing of the current bucket, since these arcs can only affect temporary distances outside the rage interval of the current bucket. As a result, they will not cause any nodes to be inserted back into the current bucket. If the current bucket remains empty after a phase, all nodes within its distance

range have been assigned their permanent distance values during the preceding phases. Consequently, all heavy arcs originating from these nodes are relaxed. After this, the algorithm searches sequentially for the next bucket that is not empty, and repeats the process. The algorithm continues until all the buckets have been processed and there are no nodes left for exploration. At this point it stops and returns the array d, where $d[v]$ contains the minimum cost to reach v from s.

The following theorem provides a complexity estimate for the sequential Delta-stepping algorithm, which serves as a fundamental component in the development of our dynamic algorithm.

Theorem 2. *([11]) The sequential delta-stepping algorithm has a time complexity of* $O(n+m+L/\Delta+n_\Delta+m_\Delta)$.

4.3 Dynamic Adjustment Single Source Shortest Path Algorithm

Let $G = (V, A, w)$ be the initial weighted digraph, where the shortest paths from the source node s to the other nodes have already been determined. Let d be the array containing the shortest path weights, and let p be the array containing the predecessors of nodes in these shortest paths. The results of running an algorithm like Dijkstra on G to find the single-source shortest paths can be represented as a shortest path tree, which is a subgraph $G' = (V, A', w)$ of G. This tree is rooted at the source node and spans all reachable nodes in the graph. The set of arcs in the tree is denoted by A', and it is defined as follows: $A' = \{(p[x], x) | p[x] \neq null\}$. Each path from the root to any other node in the tree represents the shortest path from the source to that node in the original digraph.

In the event of an arc weight change in the digraph G, whether it be an increase or a decrease, it is possible that some shortest paths may be altered, while others remain unchanged but experience an overall distance modification. Additionally, there may be some shortest paths that are not affected. The nodes in the digraph that may experience an alteration in their shortest paths, and implicitly their distances from the source, are referred to as **Affected Nodes**. The set of nodes that may only have their distance labels modified, without any change in the actual shortest path structure, are referred to as **Relabel-Only Nodes**. Finally, there are nodes that do not undergo any changes in their shortest paths or distance labels, and these can be identified as **Unaffected Nodes**.

The algorithm for updating shortest paths in response to changes in arc weights is divided into three phases.

Phase 1: Identification

In this phase, the algorithm identifies the nodes that are affected by the change in arc weight and those that only need to be relabeled. This step is important as it identifies which nodes require further processing.

Phase 2: Relabeling

In this phase, the algorithm performs some necessary distance label updates, specific to the Relabel Only Nodes or Affected Nodes, depending on the case. These updates constitute a preliminary step prior to commencing the third phase.

Phase 3: Delta-Stepping Algorithm

In this phase, the algorithm completes the computation of distance labels for the affected nodes using a modified delta-stepping algorithm. This subroutine is focused on updating the distance labels of the Affected Nodes. It maintains a list of buckets and a list of updated nodes for processing, and ensures that all potential shorter paths are evaluated, taking into account both light and heavy arcs. This targeted approach guarantees the efficient updating of the shortest paths.

The following two subsections discuss the specifics of increasing and decreasing arc weights. These are followed by a subsection in which the Dynamic Adjustment Single Source Shortest Path algorithm (DynAdjSSSPA) is presented in detail.

4.3.1 Arc Capacity Increase

Let $\widehat{G} = (N, A, \widehat{w})$ be defined as a weighted directed graph that differs from G solely in terms of the weight of an individual arc, specifically the weight of arc (k, l). In this case, the weight of arc (k, l) is greater than that of the initial weight of arc (k, l). Consequently, $\widehat{w}(x, y) = w(x, y)$ for each $(x, y) \in A \setminus \{(k, l)\}$ and $\hat{w}(k, l) = w(k, l) + \Delta w$, where Δw is a given positive amount. There are two possible cases:

Case 1: When $p[l] \neq k$ the shortest paths in G will be identical to those in $\widehat{G}$, with the same distances as in G. This means that all the nodes of $\widehat{G}$ are Unaffected Nodes.

Case 2: When $p[l] = k$ is possible for the shortest paths in the modified digraph $\widehat{G}$ to differ from those in the initial digraph G. These modified shortest paths can only be from the source node

to node l, as well as to the descendants of l in the predecessor subgraph $G' = (V, A', w)$ of G. Therefore these nodes will be the Affected Nodes and they can be identified by employing breadth-first search (BFS) within the shortest path tree $G' = (V, A', w)$. All the other nodes will be Unaffected Nodes.

4.3.2 Arc Capacity Decrease

While an increase in arc weights results in a localized update, focusing on the paths directly affected by the increase, a decrease in arc weights necessitates a more global update process. This is because the decreased weight can create new shorter paths that propagate through the network. Therefore, the decrease in an arc weight often requires a more comprehensive update.

Let $\widehat{G} = (N, A, \widehat{w})$ be the weighted directed graph that differs from G by the weight of arc (k, l) that is smaller than that of the initial weight of arc (k, l). Consequently, $\widehat{w}(x, y) = w(x, y)$ for each $(x, y) \in A \setminus \{(k, l)\}$ and $\hat{w}(k, l) = w(k, l) - \Delta w$, where Δw is a given positive amount. There are again, two possible cases:

Case 1: When $p[l] \neq k$ the the shortest paths may be altered for nodes that are reachable from the source through node l. Therefore these nodes will be the Affected Nodes and they can be identified by employing breadth-first search (BFS) within the digraph $\widehat{G} = (V, A, \widehat{w})$. All the other nodes will be Unaffected Nodes.

Case 2: When $p[l] = k$ the shortest paths in the modified digraph $\widehat{G}$ may differ from those in the initial digraph G. These modified shortest paths can only be from the source node to the nodes that are reachable from the source through node l in the graph $\widehat{G} = (V, A, \widehat{w})$, therefore these nodes will be the Affected Nodes. For node l and its descendants in the shortest path tree $G' = (V, A', w)$, the shortest paths will remain the same, and only the label distances will decrease by Δw. Therefore these nodes will be the Relabel-Only Nodes and they can be identified by employing breadth-first search (BFS) within the shortest path tree $G' = (V, A', w)$. All the other nodes, except the previous two sets, will be Unaffected Nodes.

4.3.3 Description of the Algorithm

The Dynamic Adjustment Single Source Shortest Path Algorithm (Algorithm 7) is designed to efficiently update shortest paths in a digraph when the weight of an arc changes. This

algorithm uses precomputed shortest path information, specifically the distance label vector d and the predecessor vector p, in order to implement the necessary adjustments resulting from the modification of an arc weight $w(k,l)$ by an amount equal to Δw, which can be either positive or negative.

The algorithm starts by calling the subroutine 'Affected and Relabel-Only Nodes Identification' (Algorithm 2) in order to identify those nodes which mat be affected by the change in arc weight. This subroutine performs a breadth-first search (Algorithm 1) either in the digraph G or in the predecessor subgraph G', and returns two sets: Affected Nodes and Relabel-Only Nodes. The former is comprised of those nodes whose shortest paths may be affected as a consequence of the modification to the arc weight, whereas the latter is made up of nodes that retain their shortest paths but require their distance labels to be updated.

Algorithm 1 Breadth First Search

Input: $G = (V, A)$, start node r;

Output: Reachable Set

```
1:  Queue = {r}
2:  Reachable = {r}
3:  while Queue ≠ ∅ do
4:      remove a node x from Queue
5:      for each (x, y) ∈ A do
6:          if y ∉ Reachable then
7:              Reachable = Reachable ∪{y}
8:              Queue = Queue ∪{y}
9:          end if
10:     end for
11: end while
12: return Reachable Set
```

If Δw is negative, Relabel-Only Nodes may be a non-empty set. Consequently, the algorithm updates the distance labels for the nodes in the set Relabel-Only Nodes to reflect the decreased (k,l) arc weight. If Δw is positive, the algorithm updates the distance labels for the nodes in Affected Nodes to reflect the increased (k,l) arc weight from the current paths (see lines 4-12 from Algorithm 7).

Following the update of the relabeled nodes, the algorithm proceeds to it's main subroutine which is the Delta-Stepping Modified Algorithm (Algorithm 3). This subroutine uses a modified version of the Delta-Stepping algorithm that focuses only on the nodes that are not Affected and on the Affected Nodes that have been updated. It processes each node, considering both light and heavy outgoing arcs towards Affected Nodes, to ensure that all potential shorter paths are evaluated.

The process begins with the initialization of two primary data structures: a list of buckets B and a list of sets $UpdatedNodesList$. Each bucket in B holds nodes based on their distance labels, while $UpdatedNodesList$ is employed to track nodes whose distance labels have been updated during the process. The total number of buckets is determined by computing the integer part of the maximum distance divided by the step size Δ, with the additional value of one.

Subsequently, algorithm 3 populates the initial buckets. For each node u in the graph G that is not in the set of Affected Nodes, the appropriate bucket index is calculated based on the node's current distance label : $bucket\_index = \left\lfloor \frac{d[u]}{\Delta} \right\rfloor$, where $d[u]$ is the distance label of node u. The node is then placed in the corresponding bucket.

The principal component of the algorithm is the iterative process, which continues as long as there are non-empty buckets or non-empty sets in $UpdatedNodesList$. In each iteration, the smallest non-empty bucket index i is identified. Two sets, R and $R\_updated$, are initialized to maintain records of nodes undergoing processing. Initially, the algorithm processes the non-Affected Nodes in the current bucket $B[i]$. The nodes extracted from the bucket are added one by one to the set R, after which the light arcs (arcs with weights $\leq \Delta$) leading to Affected Nodes are relaxed. This is achieved using the function $process\_light\_arcs\_to\_affected(u)$ (Algorithm 4), whereby the distance labels of the target nodes are updated if a shorter path is found. Once all non-Affected Nodes in $B[i]$ have been processed, the updated nodes in the current bucket's $UpdatedNodesList[i]$ are addressed. Each updated node is deleted from the list, then added to the set $R\_updated$, and their light arcs to Affected Nodes are similarly relaxed.

Once the light arcs for the nodes in bucket i have been fully processed, the algorithm enters the next stage, which is the processing of heavy arcs (arcs with weights > Δ). First, the function $process\_heavy\_arcs\_to\_affected(R)$ (Algorithm 5) is invoked to manage the nodes in R; then, $process\_heavy\_arcs\_to\_affected(R\_updated)$ is invoked to address the nodes in $R\_updated$. This process guarantees the evaluation of all potential shorter paths, including

Algorithm 2 Affected and Relabel-Only Nodes Identification

Input: $G = (V, A)$, $G' = (V, A')$, s, k, l, Δw, p

Output: Affected Nodes, Relabel- Only Nodes

1: Affected Nodes = $\emptyset$
2: Relabel-Only Nodes = $\emptyset$
3: **if** $\Delta w > 0$ **then**
4: **if** $p[l] = k$ **then**
5: Affected Nodes = BFS(G', l)
6: **end if**
7: **else if** $\Delta w < 0$ **then**
8: **if** $p[l] \neq k$ **then**
9: Affected Nodes = BFS(G, l)
10: **else**
11: Relabel-Only Nodes = BFS(G', l)
12: All Affected Nodes = BFS(G, l)
13: Affected Nodes = All Affected Nodes $-$ Relabel-Only Nodes $- \{s\}$
14: **end if**
15: **end if**
16: **return** (Affected Nodes, Relabel-Only Nodes)

Algorithm 3 Delta-Stepping Modified Algorithm

Input: $G = (V, A, w)$, distance labels d, predecessors p, Affected Nodes, step size Δ

Output: Updated distance label vector d, updated predecessor vector p

1: Initialize buckets B as a list of sets
2: Initialize UpdatedNodesList as a list of sets, one corresponding to each bucket
3: $num\_buckets = \lfloor \text{max distance}/\Delta \rfloor + 1$
4: **for** each node u in V $\setminus$ AffectedNodes **do**
5: $bucket\_index = \left\lfloor \frac{d[u]}{\Delta} \right\rfloor$
6: $B[bucket\_index] = B[bucket\_index] \cup \{u\}$
7: **end for**
8: **while** there exists an index i such that $B[i] \neq \emptyset$ or $UpdatedNodesList[i] \neq \emptyset$ **do**
9: $i = \min\{j : B[j] \neq \emptyset \text{ or } UpdatedNodesList[j] \neq \emptyset\}$
10: $R = \emptyset$
11: $R\_updated = \emptyset$
12: **while** $B[i] \neq \emptyset$ **do**
13: $u = B[i].\text{pop}()$
14: $R = R \cup \{u\}$
15: process_light_arcs_to_affected(u)
16: **end while**
17: **while** $UpdatedNodesList[i] \neq \emptyset$ **do**
18: $u = UpdatedNodesList[i].\text{pop}()$
19: $R\_updated = R\_updated \cup \{u\}$
20: process_light_arcs_to_affected(u)
21: **end while**
22: **if** $R \neq \emptyset$ **then**
23: process_heavy_arcs_to_affected(R)
24: **end if**
25: **if** $R\_updated \neq \emptyset$ **then**
26: process_heavy_arcs_to_affected($R\_updated$)
27: **end if**
28: **end while**

those involving arcs with higher weights.

Algorithm 4 Process Light Arcs to Affected

1: **Function process_light_arcs_to_affected(**u**)**
2: **for** each outgoing arc (u, v) such that $w(u, v) \leq \Delta$ **do**
3: **if** $v \in$ Affected Nodes **then**
4: relax($u, v, w(u, v)$)
5: **end if**
6: **end for**

Algorithm 5 Process Heavy Arcs to Affected

1: **Function process_heavy_arcs_to_affected(**R**)**
2: **for** each node u in R **do**
3: **for** each outgoing arc (u, v) such that $w(u, v) > \Delta$ **do**
4: **if** $v \in$ Affected Nodes **then**
5: relax($u, v, w(u, v)$)
6: **end if**
7: **end for**
8: **end for**

The relaxation function used in both Algorithm 4 and Algorithm 5 it is a modified relaxation function (Algorithm 6). This version of the relaxation function extends the classic relaxation operation by incorporating bucket management to optimize the processing of nodes. The function is responsible for computing a new potential distance for a target node v via an arc (u, v). If the newly computed distance is shorter than the actual distance label, the function updates the distance label and marks the node as updated, inserting it into the corresponding list of updated nodes, according to its new distance.

The Dynamic Adjustment Single Source Shortest Path Algorithm ends by returning the optimal shortest paths from the source node to all other nodes in the modified digraph. These paths can be reconstructed from the updated distance label vector d and the updated predecessor vector p, which represents the output of the algorithm. This dynamic adjustment mechanism ensures that recalculation is efficient, without the necessity of recomputing the entire shortest path tree from scratch.

Algorithm 6 Relax Function with Bucket Check

1: **Function relax(**u, v, w**)**
2: $new\_distance = d[u] + w$
3: **if** $new\_distance < d[v]$ **then**
4: $current\_bucket\_index = \left\lfloor \frac{d[v]}{\Delta} \right\rfloor$
5: $new\_bucket\_index = \lfloor \frac{new\_distance}{\Delta} \rfloor$
6: $found =$ **false**
7: **for** each bucket j in UpdatedNodesList **do**
8: **if** $v \in UpdatedNodesList[j]$ **then**
9: $found =$ **true**
10: **if** $current\_bucket\_index \neq new\_bucket\_index$ **then**
11: Remove v from $UpdatedNodesList[j]$
12: $UpdatedNodesList[new\_bucket\_index] = UpdatedNodesList[new\_bucket\_index] \cup \{v\}$
13: **end if**
14: **break**
15: **end if**
16: **end for**
17: **if not** $found$ **then**
18: $UpdatedNodesList[new\_bucket\_index] = UpdatedNodesList[new\_bucket\_index] \cup \{v\}$
19: **end if**
20: $d[v] = new\_distance$
21: $p[v] = u$
22: **end if**

Algorithm 7 Dynamic Adjustment Single Source Shortest Path Algorithm

Input: Digraph $G = (V, A, w)$, distance label vector d, predecessor vector p, arc (k, l), weight change Δw

Output: Updated distance label vector d, updated predecessor vector p

Phase 1: Find Affected and Relabel Only Nodes
(Affected Nodes, Relabel-Only Nodes) =
    = Affected and Relabel-Only nodes Identification$(G, G', (k, l), \Delta w, p)$
Phase 2: Relabel Nodes
if $\Delta w < 0$ **then**
    for each node u in Relabel-Only Nodes **do**
        $d[u] = d[u] + \Delta w$
    end for
end if
if $\Delta w > 0$ **then**
    for each node u in Affected Nodes **do**
        $d[u] = d[u] + \Delta w$
    end for
end if
Phase 3: Update distances. Call Delta-Stepping Modified Algorithm
(d, p) = Delta-Stepping Modified Algorithm$(\widehat{G}, d, p,$ Affected Nodes$)$
return (d, p)

4.4 Conclusions

This chapter presents the Dynamic Adjustment Single Source Shortest Path Algorithm (DynAdjSSSPA), which uses a modified version of the Delta-stepping algorithm to efficiently manage shortest path computations in dynamic graph environments. Our investigation into the algorithm's correctness and complexity highlights several potential findings and areas for future research and improvements.

In order to ensure the correctness of the DynAdjSSSPA, it is important to identify and process all affected nodes whose shortest paths may change due to arc weight modifications. This is achieved through a structured approach involving the identification of affected nodes (phase 1), relabelling of nodes (phase 2), and the application of the Delta-stepping algorithm for final adjustments (phase 3). The correctness of the algorithm is supported by its capability to correctly differentiate between affected nodes, relabel only nodes and unaffected nodes, thereby ensuring accurate and efficient updates to the shortest paths.

The use of buckets in the Delta-stepping algorithm allows for the organized processing of nodes based on their distance ranges. Despite the worst-case bounds favoring label-setting algorithms like Dijkstra's, the practical performance of label-correcting algorithms, such as Delta-stepping, is often superior for large graphs with random arc weights.

Chapter 5

Flow Increment through Network Expansion

5.1 Introduction

In modern infrastructure management, network expansion is a critical task to meet the growing demand for essential services such as electricity, gas, water and data. This expansion can be achieved by increasing the transport capacity of existing wires, pipes or bandwidths. Alternatively, new connections can be added to the system. However, each of these strategies is constrained by physical or technological limitations and comes at a significant cost. Given these constraints, the challenge is to optimize network design to achieve maximum efficiency at minimum cost.

The field of network flow optimization offers a number of problem formulations that are directly related to these challenges. Two such problems, which are central to the discussions in this chapter, are **the inverse maximum flow problem** and **the reverse maximum flow problem**. These problems use sophisticated mathematical models to find optimal solutions and involve strategic changes in the capacity of network arcs to meet specific objectives.

The inverse maximum flow problem provides a mathematical framework for making minimal adjustments to the existing capacities of a network in order to achieve the desired maximum flow, while minimizing the changes to the original capacities. The inverse maximum flow problem arose from the need to optimize existing network infrastructures without the extensive costs associated with building new connections or completely redesigning the net-

work. This problem is particularly relevant in contexts where physical, financial or regulatory constraints limit the possibility of expanding the capacity of the network by traditional means. The optimization of the network is quantified using various norms to measure capacity adjustments. In particular, the problem under the L_∞ norm can be solved efficiently using a binary search approach, as shown in [4]. When considering the L_k norm, a strongly polynomial algorithm has been developed that mainly involves computing a minimum cut in a special network [5]. Furthermore, the problem has been shown to be NP-hard [21], and its complexity increases in scenarios involving losses or gains on arcs.

The reverse maximum flow problem [22] aims to identify new capacity vectors that ensure the maximum flow in a modified network is maintained above a specified threshold, v_0, while reducing the distance, measured by the Chebyshev norm, between the initial and the new capacity vectors. A polynomial algorithm, which operates in two phases, has been developed to solve this problem. In the initial phase, a binary search is employed to identify an interval that contains the optimal value of the flow, v_0. Subsequently, in the second phase, the Newton method, as described in [24] , is employed to determine the optimal capacity vector. The Reverse Maximum Flow problem is particularly relevant in contexts where it is critical to control or reduce network capacity without completely redesigning the network, such as during downsizing operations, or to comply with new environmental or safety regulations. Applications include environmental management to prevent overuse of resources, ensuring safety compliance in the chemical, oil and gas industries, and managing network capacity during economic slowdowns or corporate reorganization.

The problem of network expansion under budgetary constraints has been the studied in [7]. The objective is to make use of the budget in an optimal manner, while maximizing the flow within the network, given a budget, a maximum potential expansion of arc capacity and an arc capacity expansion cost function. This is referred to as the budget-constrained flow expansion problem (BFEP). The fundamental approach to solving this problem is to incrementally increase the flow in the network by one unit, using the most cost-effective path from the source to the sink. At each iteration, the remaining budget is recalculated, and the process concludes when the entire budget has been expended. The solution to this problem is based on two polynomial algorithms. One algorithm identifies the arcs that require extension to achieve a one-unit increase in flow at minimal cost. The second iteration calls the first and verifies whether the imposed budget limit has been reached. The Budget-Constrained Flow Expansion Problem (BFEP) effectively incorporates strategies from both the inverse maximum flow problem and

the reverse maximum flow problem to improve its approach to optimising network expansion within fixed budget constraints. The connection with the inverse maximum flow problem is obvious, as both focus on maximising network flow - BFEP does so within the constraints of the budget, while the inverse problem seeks to achieve this through minimal capacity adjustments. This common goal allows BFEP to use strategies from the inverse problem to determine which changes will most efficiently improve flow. Similarly, the relationship with the reverse maximum flow problem is critical in guiding capacity management. The reverse problem, which typically aims to reduce capacity to avoid exceeding certain flow thresholds, provides valuable insights into managing expansions to ensure they are necessary and sustainable. In the same way that the reverse problem uses capacity adjustments to maintain operational efficiency and safety, the BFEP applies these principles to ensure that any expansion is both economically viable and essential to meet the demands of the network, thereby ensuring strategic and cost-effective growth of the network.

A wide range of applications of the maximum flow problem can be found in the literature, such as transport problems [20], [2] and pipeline transport problems [9], [10], [23], and in all of these types of problems, network expansion may be necessary to increase the amount of flow transported through the network. This necessity provides a compelling background for the analysis that is carried out in this chapter, in which we deal with the minimum cost network expansion problem (MCNEP).

The objective of MCNEP, beginning with a given network G, is to obtain a new network G', by increasing the capacities of the arcs from G within defined limits, or by adding new arcs, in order to achieve the lowest possible modification cost while enabling w units of flow to be transported from source to sink. The cost associated with capacity augmentation and arc insertion is a linear function with respect to the modification of respective capacities. Our research is guided by the practical requirements of applying this problem in the field of infrastructure, ensuring that changes, whether to increase capacity or to add new arcs, are both cost-effective and substantial enough to meet the specified flow requirements. This focus reinforces the relevance of our study and highlights the broader implications of MCNEP solutions in real-life network management scenarios.

The results from this chapter were published in [6] and their presentation in this chapter of the thesis is organized as follows. The Minimum Cost Network Expansion Problem (MCNEP) is formally described in Section 5.2 and a strongly polynomial algorithm for solving this problem is introduced. Section 5.3 concludes this work and identifies future potential research

directions.

5.2 Flow increment through Network Expansion

As previously stated, the Minimum Cost Network Expansion Problem (MCNEP) is concerned with the strategic expansion of a given network G into a new network G by increasing the capacities of existing arcs or by adding new arcs within defined limits, to ensure that the modified network G can allow w units of flow to pass from source to sink while minimising the modification costs. The cost of the network modification is a linear function with respect to the changes in capacity. The economic aspects of this network expansion is addressed by introducing a cost function $c : A \to \mathbb{R}^*_+$, where $c(a)$ is the cost of modification of the capacity on the arc $a \in A$ per unit. Thus, if the capacity of the arc a increases by d units, the overall cost of changing the capacity of a will be $c(a) \cdot d$. Furthermore, in order to avoid over-expansion of the network capacity, an upper bound modification function $\alpha : A \to \mathbb{R}^*_+$ is imposed, setting $u(a) + \alpha(a)$ as the maximum allowable capacity for each arc a.

Let Q be the set of arcs that can be added to the network. Of course, $Q \subseteq V \times V$, and $Q \cap A = \phi$, i.e., $Q \subseteq V \times V - A$. A cost function $c_Q : Q \to \mathbb{R}^*_+$ is also introduced, where $c_Q(a)$ is the per unit cost of the capacity if the arc $a \in Q$ is added to the network. Therefore, if the arc $a \in Q$ having the capacity $u_Q(a) > 0$ is introduced into the network, then the cost of adding a to the network is $c_Q(a) \cdot u_Q(a)$. An upper limit function $\beta : Q \to \mathbb{R}^*_+$ for the capacities of the arcs in Q is also introduced, where $\beta(a)$ is the maximum allowed capacity for the arc $a \in Q$ if it is introduced into the network, i.e., $u_Q(a) \leq \beta(a)$, where $u_Q(a) > 0$ is the capacity of a.

The objective is to identify **the minimum cost expansion of the network**, G, by increasing the capacities of the arcs and by adding new arcs, such that in the resulting network G', w units of flow can be transported from s to t. That is, there exists a feasible flow of value w in G'. Therefore the following problem must be solved:

$$\begin{cases} min\{\sum_{a\in A}(c(a)\cdot(v(a)-u(a))+\sum_{a\in Q}(c_Q(a)\cdot u_Q(a))\} \\ u(a)\leq v(a)\leq u(a)+\alpha(a), \forall a\in A \\ 0\leq u_Q(a)\leq \beta(a), \forall a\in Q \\ \text{there exists a feasible flow of value } w \text{ in } G'=(V,A',s,t,u') \\ A'=A\cup\{a\in Q \mid u_Q(a)>0\} \\ u'(a)=\begin{cases} v(a), a\in A \\ u_Q(a), a\in A'-A \end{cases}, \forall a\in A' \end{cases} \tag{5.1}$$

We shall name the problem from Eq. (5.1) as the *minimum cost network expansion problem*, and denote it as MCNEP. We can formulate the following result:

Theorem 3. *If $v(f^*)\geq w$, where f^* is the maximum flow in the network G, then G is the solution of MCNEP.*

Proof. Let f^* be the maximum flow in the network G. We assume that $v(f^*)\geq w$. It follows that there exists a feasible flow f in G so that $v(f)=w$. So, G is a feasible solution of MCNEP. Since the cost of the objective function in Eq. (5.1) for G is 0, it is obvious that G represents the optimal solution for MCNEP. □

We shall now investigate the feasibility of MCNEP. It can be shown that the maximum value of the flow that can be transported from s to t occurs when the capacities of all the arcs from A are increased to their maximum value and all the arcs from Q are incorporated into the network, with the maximum allowable capacities being used. It therefore follows that the maximum value of flow that can be transported from s to t is obtained in the network $G''=(V,A\cup Q,s,t,u'')$, where:

$$u''(a)=\begin{cases} u(a)+\alpha(a), a\in A \\ \beta(a), a\in Q \end{cases} \tag{5.2}$$

We shall call G'' as the *maximum extended network* since all the capacities of the arcs from $A\cup Q$ are set to their maximum.

We have the following feasibility theorem for MCNEP:

Theorem 4. *MCNEP is feasible if and only if $v(g^*)\geq w$, where g^* is a maximum flow in G''.*

Proof. Let's assume that MCNEP is feasible and $v(g^*) < w$, where g^* is a maximum flow in G''. Since MCNEP is feasible, it follows that there is a network $G' = (V, A', s, t, u')$ so that $u(a) \leq u'(a) \leq u(a) + \alpha(a), \forall a \in A$, $0 \leq u'(a) \leq \beta(a), \forall a \in Q$, and there is a feasible flow f' in G' so that $v(f') = w$. From (5.2) we may conclude that f' is a feasible flow in G'', and since f^* is a maximum flow in G'' it follows that $v(g^*) \geq v(f') = w$, in contradiction with the initial assumption that $v(g^*) < w$.

Now, for the inverse implication, we suppose that for the maximum flow g^* in G'' we have $v(g^*) \geq w$. It results that there is a feasible flow f'' in G'' so that $v(f'') = w$. So, G'' is a feasible solution for Eq. (5.1). □

If $v(f^*) < w$, with f^* being maximum flow in the network G (G is not the solution of MCNEP, see Theorem 3) and MCNEP passes the feasibility test given by Theorem 4,this means that a solution for MCNEP exists and has to be found. To do that, we create a new network denoted $G^e = (V^e, A^e, s, t, u^e, c^e)$. The set V^e contains all the nodes from V, and for each arc $a \in A$, a new node denoted i_a is introduced in V^e, i.e.,

$$V^e = V \cup V_A, V_A = \{i_a \mid a \in A\} \tag{5.3}$$

The set A^e contains all the arcs from $A \cup Q$, and for each arc $a = (i, j) \in A$ two new arcs are added, (i, i_a), and, respectively, (i_a, j), i.e.,

$$A^e = A \cup Q \cup A_1^e \cup A_2^e, \tag{5.4}$$

where:

$$A_1^e = \{(i, i_a) \mid a = (i, j) \in A\}, A_2^e = \{(i_a, j) \mid a = (i, j) \in A\} \tag{5.5}$$

The capacity function u^e is defined as follows:

$$u^e(i,j) = \begin{cases} u(i,j), \text{if } (i,j) \in A \\ \beta(i,j), \text{if } (i,j) \in Q \\ \alpha(a), \text{if } j = i_a \text{ or } i = i_a \text{ where } i_a \in V_A \end{cases} \tag{5.6}$$

The cost function c^e is defined as:

$$c^e(i,j) = \begin{cases} 0, \text{if } (i,j) \in A \\ c_Q(i,j), \text{if } (i,j) \in Q \\ c(a)/2, \text{if } j = i_a \text{ or } i = i_a \text{ where } i_a \in V_A \end{cases} \tag{5.7}$$

Let f^e be a minimum cost flow of value w in G^e, meaning that, f^e is a feasible flow of value w in G^e that has the minimum cost among all feasible flows of value w in G^e, where the cost of a feasible flow f in G^e denoted $c^e(f)$ is defined as:

$$c^e(f) = \sum_{a \in A^e} c^e(a) \cdot f(a) \tag{5.8}$$

We consider the network denoted $G^* = (V, A^*, s, t, u^*)$ having:

$$A^* = A \cup A^{*'}, A^{*'} = \{a \in Q \mid f^e(a) > 0\} \tag{5.9}$$

and

$$u^*(a) = \begin{cases} u(a) + f^e(i, i_a), \text{if } a \in A \\ f^e(a), \text{if } a \in A^* - A \end{cases} \tag{5.10}$$

Theorem 5. *The network $G^* = (V, A^*, s, t, u^*)$ defined using Eq. (5.9) and Eq. (5.10) is the optimum solution of MCNEP.*

Proof. Remembering that G^* is constructed using a minimum cost flow f^e of value w calculated in the network $G^e = (V^e, A^e, s, t, u^e, c^e)$, it means that f^e is the solution of the problem:

$$\begin{cases} min\{\sum_{a \in A^e} c^e(a) \cdot f(a)\} \\ f \text{ is a feasible flow of value } w \text{ in } G^e \end{cases} \tag{5.11}$$

Therefore:

$$\begin{aligned} &\sum_{a \in A^e} c^e(a) f^e(a) = \\ &= \sum_{a \in A} c^e(a) f^e(a) + \sum_{a \in Q} c^e(a) f^e(a) + \sum_{a \in A_1^e} c^e(a) f^e(a) + \sum_{a \in A_2^e} c^e(a) f^e(a) = \\ &= \sum_{a \in Q} c_Q(a) f^e(a) + \sum_{a=(i,j) \in A} c^e(i, i_a) f^e(i, i_a) + \sum_{a=(i,j) \in A} c^e(i_a, j) f^e(i_a, j) = \\ &= \sum_{a \in Q} c_Q(a) f^e(a) + \sum_{a=(i,j) \in A} c(a)/2(f^e(i, i_a) + f^e(i_a, j)) = \\ &= \sum_{a \in Q} c_Q(a) f^e(a) + \sum_{a=(i,j) \in A} c(a) f^e(i, i_a) = \\ &= \sum_{a \in Q} c_Q(a) u^*(a) + \sum_{a \in A} c(a)(u^*(a) - u(a)). \end{aligned} \tag{5.12}$$

Using Eq. (5.11) and Eq. (5.12) we get the following optimization problem:

$$min\{\sum_{a \in A} c(a) \cdot (u^*(a) - u(a)) + \sum_{a \in Q} c_Q(a) \cdot u^*(a)\} \tag{5.13}$$

Using Eq. (5.10), we have:

$$0 \leq u^*(a) - u(a) = f^e(i, i_a) \leq \alpha(a), \forall a \in A \tag{5.14}$$

$$0 < u^*(a) = f^e(a) \leq \beta(a), \forall a \in Q \text{ and } f^e(a) > 0 \tag{5.15}$$

$$0 = u^*(a) \leq \beta(a), \forall a \in Q, \text{ and, we agree that, } f^e(a) = 0. \tag{5.16}$$

We consider the following flow denoted f^* in G^*:

$$f^*(a) = \begin{cases} f^e(a) + f^e(i, i_a), \text{if } a \in A \\ f^e(a), \text{if } a \in A^* - A \end{cases} \tag{5.17}$$

Since f^e is a feasible flow in G^e, we can conclude that f^* satisfies the conservation conditions in G^*, and from Eq. (5.10) it follows that f^* respects the boundary conditions in G^*. That means that f^* is a feasible flow in G^*. Using Eq. (5.13-5.16) we may conclude that u^* is the optimum solution of Eq. (5.1).

□

Corollary 5.1. *The cost of network expansion from G to the optimum solution G^* of MCNEP is $v(f^e)$, where f^e is the minimum cost flow in G^e.*

Proof. The result is immediate from Eq. (5.12). □

Using Theorem 3, Theorem 4, and Theorem 5, the following algorithm (Algorithm 8) for solving MCNEP is developed.

Theorem 6. *The time complexity of algorithm 8 (AMCNEP) is*

$$O((m+q)^2 \cdot \log n + (m+q) \cdot m \cdot \log^2 n)$$

where n is the number of vertices in G, m is the number of arcs of the network G, and q the number of arcs from Q (that can be added to the network G).

Proof. Algorithm 8 has to compute once the maximum flow in G, and once the maximum flow in G''. Today's most efficient algorithm for the maximum flow problem is the one proposed by Orlin [15]. It has a time complexity of $O(m \cdot n)$ when is applied in the original network G. In G'' there are n still vertices but $m+q$ arcs. Therefore in this case, maximum flow can be computed in $O((m+q) \cdot n)$.

Algorithm 8 Algorithm for solving MCNEP (AMCNEP)

Input: $G = (V, A, s, t, u)$, c, α, β, Q, c_Q, w

Output: $G^* = (V, A^*, s, t, u^*)$

1: Find a maximum flow f^* in G
2: **if** $v(f^*) \geq w$ **then**
3: Is not necessary to modify G, since it can accomodate a feasible flow of value w.
4: **return**
5: **end if**
6: Build the network G'' using Eq. (5.2)
7: Find a maximum flow g^* in G''
8: **if** $v(g^*) < w$ **then**
9: MCNEP is not feasible
10: **return**
11: **end if**
12: Build the network $G^e = (V^e, A^e, s, t, u^e, c^e)$ using Eq. (5.3) - (5.7)
13: Find a minimum cost flow f^e in G^e
14: Build the network $G^* = (V, A^*, s, t, u^*)$ using f^e, Eq. (5.9) and Eq. (5.10)
15: The network G^* is the optimum solution for MCNEP

In G^e we have to compute the minimum cost flow. Minimum cost flow are assumed to work on integer values capacity networks [1]. Since in G^e the cost of the arcs from $A_1^e \cup A_2^e$ are integer values divided by 2 (see Eq. (5.7), before proceeding to the computation of the minim cost flow, all the costs of the arcs from A^e are multiplied by 2, to ensure they have integer valus, and, in the end, the overall cost of the obtained flow f^e it will be divided by 2. The most efficient algorithm that is known today for computing minimum cost flow is also due to Orlin [14]. Since the network G^e has $m + n$ nodes and $3m + q$ arcs, if the algorithm is applied in G^e, then it has a time complexity of $O((m + q)^2 \cdot \log n + (m + q) \cdot m \cdot \log^2 n)$.

Therefore, the overall time complexity of the proposed algorithm is $O((m + q)^2 \cdot \log n + (m + q) \cdot m \cdot \log^2 n)$. □

5.3 Conclusions

This chapter has investigated in detail the Minimum Cost Network Expansion Problem (MCNEP), an important problem that arises when existing networks need to be adapted to support increased flow from source to sink nodes. The problem is complex and involves either increasing the transport capacity of existing arcs or integrating new arcs into the network. A key aspect of MCNEP is the optimization of the costs associated with these modifications. To address this complex challenge, a strongly polynomial algorithm (Algorithm 8) has been developed and presented here, which provides a systematic approach to achieving cost-effective network expansion.

An interesting direction for future research could be a generalisation of MCNEP. Future studies could explore scenarios where the costs associated with changing arc capacities and introducing new arcs do not follow a linear model. This extension of the problem framework could include cost functions that are perhaps polynomial, exponential, or based on other non-linear models that may more accurately reflect the real world costs of network changes. This generalisation could significantly improve the applicability and relevance of the MCNEP framework to a wider range of real-world situations where cost dynamics are more complicated and varied.

Chapter 6

Integration of Remote Sensing Data with Dynamic Network Processing

6.1 Introduction

Remote sensing is the acquisition of information about an object or phenomenon without direct physical contact. Technologies such as multispectral (MS) and hyperspectral (HS) imaging have significantly transformed the methods used to observe and analyze the Earth's surface. Combining these imaging techniques with dynamic graph algorithms merges two advanced technologies, offering innovative solutions to complex, real-world problems.

MS and HS images capture information by sensing electromagnetic radiation. The main difference between them is their spectral resolution: HS images capture data in several narrow, contiguous bands, while MS images capture data in fewer, discrete bands. This detailed data allows precise identification and analysis of materials, enhancing our ability to monitor and manage natural and human-made environments. Applications range from environmental monitoring and agriculture to medical diagnostics and urban planning.

Dynamic networks and associated algorithms are essential for problems where the underlying network structure changes over time. They can be an important tool in a number of environmental applications or urban planning scenarios.

MS and HS images provide highly accurate environmental data, which can be used to construct dynamic networks. This integration allows for precise identification of nodes and edges and the assignment of appropriate weights and capacities. Consequently, combining

these imaging techniques with dynamic network algorithms opens new possibilities for addressing complex problems with temporal and spatial dimensions. For example, spectral data in environmental monitoring can help map regions and their conditions into a dynamic network, enabling real-time adjustments based on current environmental changes.

The present chapter is based on previous research into into MS and HS image processing [17], [16], [18]. The objective is to combine these sophisticated imaging techniques with dynamic graph theory to enhance the capabilities and applications of both technologies, effectively addressing real-world challenges.

6.2 Fundamentals of MS and HS Imaging

The perception of objects by the human eye is a result of the reflection of light from the objects in question. This reflected light reaches the eye, where it is processed by the brain to create a visual image. The light is a form of electromagnetic radiation, which propagates as both electric and magnetic waves in packets of energy known as photons.

The electromagnetic spectrum is a continuous range of electromagnetic radiation, categorized by wavelength. It is divided into bands, each designated for specific types of radiation within certain wavelength intervals. The electromagnetic spectrum spans a vast range of electromagnetic radiation, with wavelengths spanning from very short gamma rays to very long radio waves (see Figure 6.1). Each band of the spectrum serves different scientific, technological, and practical applications.

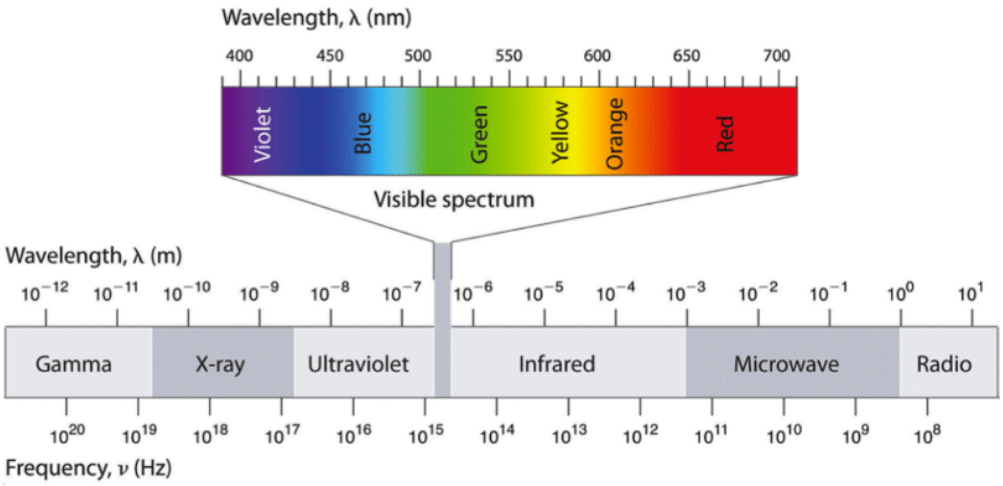

Figure 6.1: Illustration of the electromagnetic spectrum. Image courtesy of [25]

The significance of the spectral bands lies in the fact that materials reflect and absorb

light in unique ways throughout the spectrum, allowing their identification based on spectral signatures. The reflection of light across various spectral bands enables to differentiate the material condition, vegetative health, and the surface composition.

The principles of the electromagnetic spectrum form the foundation for MS and HS imaging technologies. These technologies use specific wavelength bands to capture detailed information beyond conventional visible light imaging, offering significant insights into a wide range of applications.

MS imaging is defined as an imaging technology that uses the spectral characteristics and properties of light to capture data across a range of discrete wavelength bands, typically up to 20 , across the electromagnetic spectrum. As opposed to the traditional RGB approach, which employs three broad bands, MS imaging makes use of narrower and more numerous bands, extending from the ultraviolet to the near-infrared regions. This approach enables the detailed analysis of materials by providing calibrated reflectance data, which is essential for applications such as agriculture, environmental monitoring, and medical diagnostics.

HS imaging involves the acquisition of data over a continuous spectrum, with the recording of hundreds of narrow, contiguous bands for each pixel within the image. This high spectral resolution allows for the detection of even subtle differences in spectral signatures, rendering HS imaging an optimal choice for a range of applications requiring detailed spectral information, including remote sensing, medical diagnostics, and food quality analysis. Although more complex and time-consuming than MS images, HS images provides comprehensive spectral profiles for each pixel, offering a deeper understanding of the materials being studied.

The application of MS and HS imaging is extensive and diverse, with numerous practical applications in various fields. These include agriculture, environmental research, disaster management, and more. While HS imaging is highly sensitive to environmental conditions and requires thorough calibration, making it most suitable for controlled settings or specific scientific studies, it is particularly valuable in geology and mineral development for gathering detailed information about materials. By contrast, MS imaging is relatively less constrained by the vicissitudes of the environment and atmospheric interference; it is therefore more suitable for a broader spectrum of environments and applications. The use of MS images in agriculture and forestry is extensive. Its applications include the gathering of data concerning the surface and cover of Earth, as well as the patterns of change observed therein. MS imaging is a more readily accessible and practical option for general remote sensing requirements, with the majority

of satellites and constellations readily providing MS data. Consequently, while HS imaging has considerable unexploited potential, MS imaging is currently sufficient for the needs of casual users across various remote sensing applications.

6.3 Real-Time Agricultural Monitoring Using Dynamic Networks

The integration of dynamic networks with remote sensing data represents a significant advancement in a series of activities related to Earth monitoring and environmental management. Based on the capabilities of remote sensing technologies, such as MS and HS imaging, in conjunction with dynamic graph processing, it is possible to achieve more precise and timely insights into various natural and man-made phenomena. This thesis focuses on a case study example of the use of remote sensing techniques in the agricultural sector. This choice is motivated by the particular characteristics of the available data set, which comprises a set of MS images predominantly of agricultural land with an urban component.

Given the rising global demand for agricultural resources, agriculture represents an important area of study and research. This demand extends to food production, bio-fuels, and other essential agricultural outputs that sustain humanity. With population growth, the need for more efficient and sustainable agricultural practices intensifies. Furthermore, the economic significance of agriculture in numerous regions accentuates the necessity for innovative strategies that can improve productivity and sustainability.

It is of significant importance to recognize the considerable variability in soil properties that characterize agricultural fields. Such variability can exert a considerable influence on crop health and yield. The soil characteristics, including nutrient content, pH, moisture levels and microbial activity, can vary across different parts of a field. Consequently, the uniform application of fertilizers and pesticides can result in a number of adverse effects:

- It has been proven that the use of heavy machinery on agricultural land can lead to the compaction of the soil, which disrupts its structure. This process reduces the soil's porosity to a degree that limits air circulation and the infiltration of water. These factors are crucial in enabling healthy root growth and the uptake of nutrients by crops.
- The soil is home to various microorganisms that hace an active role in nutrient cycling and in the decomposition of organic matter. Excessive use of chemicals and mechanical disruption can harm these microbial communities, reducing soil fertility and health.

In light of these factors, it can be concluded that localized interventions are more beneficial than blanket applications of fertilizers or pesticides. The targeted application of fertilizers and pesticides minimizes soil disturbance and preserves the natural microbial activity essential for sustainable agriculture. Furthermore, by applying fertilizers and pesticides only where needed, farmers can significantly reduce the overall chemical usage, lowering the risk of environmental contamination and preserving local biodiversity. This approach helps to protect species within the trophic chain from harmful chemicals. The minimization of the utilization of heavy machinery and chemical agents also serves to reduce the carbon footprint associated with agricultural operations.

The dynamic network model presented in this chapter of the thesis, integrated with real-time multispectral image processing, enables the precise monitoring and management of agricultural fields. To implement this model, we use a Dynamic Vegetation Monitoring Framework (DVMF). This framework involves several key steps:

1. **Real-time data aquisition**. The acquisition of multispectral images in real-time allows for the timely identification and monitoring of any changes in the vegetation health and soil conditions. The real-time data is of extremely important for the accurate and up-to-date construction of the dynamic network.

2. **Dynamic graph construction**. The construction of a dynamic graph is a further key step in this process. In this graph, nodes represent vegetated areas and edges denote relationships such as proximity or shared irrigation systems. This stage comprises the preliminary construction of the graph and its subsequent continuous updating, as will be detailed in the algorithm.

3. **Localised intervention**. The dynamic network processing enables farmers to identify specific areas requiring intervention, whether for irrigation, fertilization, or pest control. This promotes efficient resource use and minimizes environmental impact. The framework's change detection and alert mechanisms ensure that interventions are timely and targeted.

The following subsections will present a detailed account of the dataset and of the framework, accompanied by step-by-step illustrations of the examples generated using the available dataset, and a set of particular vegetation indices for measuring vegetation health, along with a particular metric for measuring unwanted change in vegetation status. These sections will

emphasize the practical application of the algorithm, demonstrating the tangible advantages and implementation of this approach.

In addition, available in-situ measurements integrated with the dynamic network would increases the precision of monitoring and decision-making processes, ultimately leading to more favorable agricultural outcomes. This approach supports the goal of sustainable agriculture by ensuring that interventions are targeted and based on real-time data, thereby reducing waste and minimizing adverse environmental effects.

6.3.1 Dynamic Vegetation Monitoring Framework

The Dynamic Vegetation Monitoring Framework is designed to process real-time multispectral images and maintain the dynamic graph necessary for effective monitoring and localized interventions. This framework can be used for any type of change monitoring in vegetation status. Depending on the characteristics of the vegetation, other measurements (vegetation indices) may need to be computed and used, and the targeted interventions have to be customized. Below are the detailed steps of the framework:

Algorithm 9 Dynamic Vegetation Monitoring Framework (DVMF) - Part 1

Input: Real-time multispectral images, a set of vegetation indices used as a measure of vegetation health
Output: Initial dynamic graph of vegetation monitoring
Step 1: Initial Data Acquisition
image ← `AcquireAndPreprocessImage`()
indices ← `ComputeVegetationIndices`(image)
Step 2: Graph Initialization
zones ← `IdentifyRelevantZones`(indices)
for each zone in zones **do**
    Create a node with attributes: {set of corresponding vegetation indices, spatial information (pixel coordinates or segment size)}
end for
graph ← `ConstructInitialGraph`(zones)
`EstablishEdges`(graph)

The detailed steps and processes involved in the Dynamic Vegetation Monitoring Framework (DVMF) are illustrated in Figure 6.2. This figure provides a visual representation of the

Algorithm 10 IdentifyRelevantZones

Input: Vegetation indices
Output: Zones with relevant vegetation
Identify and delimit zones within the large area of interest that have relevant vegetation for monitoring using a threshold on the relevant vegetation index. These zones can represent individual pixels or aggregated areas (segments) with similar vegetation characteristics.
Return zones

Algorithm 11 ConstructInitialGraph

Input: Zones with relevant vegetation
Output: Initial dynamic graph
Construct the initial graph with nodes representing vegetated areas (either pixels or aggregated zones) that are to be monitored
Return graph

Algorithm 12 EstablishEdges

Input: Graph
Establish edges between adjacent nodes (pixels or zones) based on available information, such as spatial relationships, similarity relationships, shared irrigation systems, soil types and quality similarity etc.

Algorithm 13 Dynamic Vegetation Monitoring Framework (DVMF)

Input: Real-time multispectral images, vegetation indices, and criteria for changes
Output: Updated dynamic graph with alerts
Step 3: Real-time Monitoring Loop
while new multispectral image is available **do**
    image ← `AcquireAndPreprocessImage`()
    indices ← `ComputeVegetationIndices`(image)
    graph ← `CopyOldGraph`()
    `UpdateNodes`(graph, indices)
    `UpdateGraph`(graph, indices)
    `SegmentAndAnalyze`(graph)
    `VisualizeAndReport`(graph)
    `RespondToAlerts`()
    `LogChanges`()
end while

Algorithm 14 AcquireAndPreprocessImage

Output: Preprocessed multispectral image
Acquire the multispectral image
Perform georeferencing, cloud masking, and atmospheric correction
Return preprocessed image

Algorithm 15 ComputeVegetationIndices

Input: Preprocessed multispectral image
Output: Vegetation indices
Compute vegetation indices from the image (e.g., NDVI, NDWI)
Return indices

Algorithm 16 CopyOldGraph

Output: Copy of the old graph
Copy the previous state of the dynamic graph
Return copied graph

Algorithm 17 UpdateNodes

Input: Graph, Vegetation indices
for each node in the graph **do**
    Update vegetation indices
    if change indicates vegetative stress **then**
        Tag node for alert analysis
    else
        Remove the node
    end if
end for

Algorithm 18 UpdateGraph

Input: Graph, Vegetation indices
for each point within the area of interest and not in the old graph **do**
    Compute vegetation indices
    if index above threshold **then**
        Add the point as a new node
    end if
end for

Algorithm 19 SegmentAndAnalyze

```
Input: Graph
Segment nodes tagged for alert analysis based on shared characteristics (e.g., similar indices, proximity) to form larger zones
for each zone do
    if zone exhibits significant unwanted change then
        Trigger an alert
    end if
end for
```

Algorithm 20 VisualizeAndReport

```
Input: Graph
Generate visualizations of the dynamic graph, highlighting nodes and zones with alerts
Create a report of nodes and zones with alerts and recommended actions
```

Algorithm 21 RespondToAlerts

```
for each alert in the report do
    Respond with specific actions (e.g., adjust irrigation, apply fertilizers)
    Monitor intervention effectiveness and log changes for trend analysis
end for
```

Algorithm 22 LogChanges

```
Log changes over time for analysis and prediction
```

framework, highlighting the sequence of operations from initial data acquisition to the real-time monitoring loop, including the segmentation, analysis, and response to alerts.

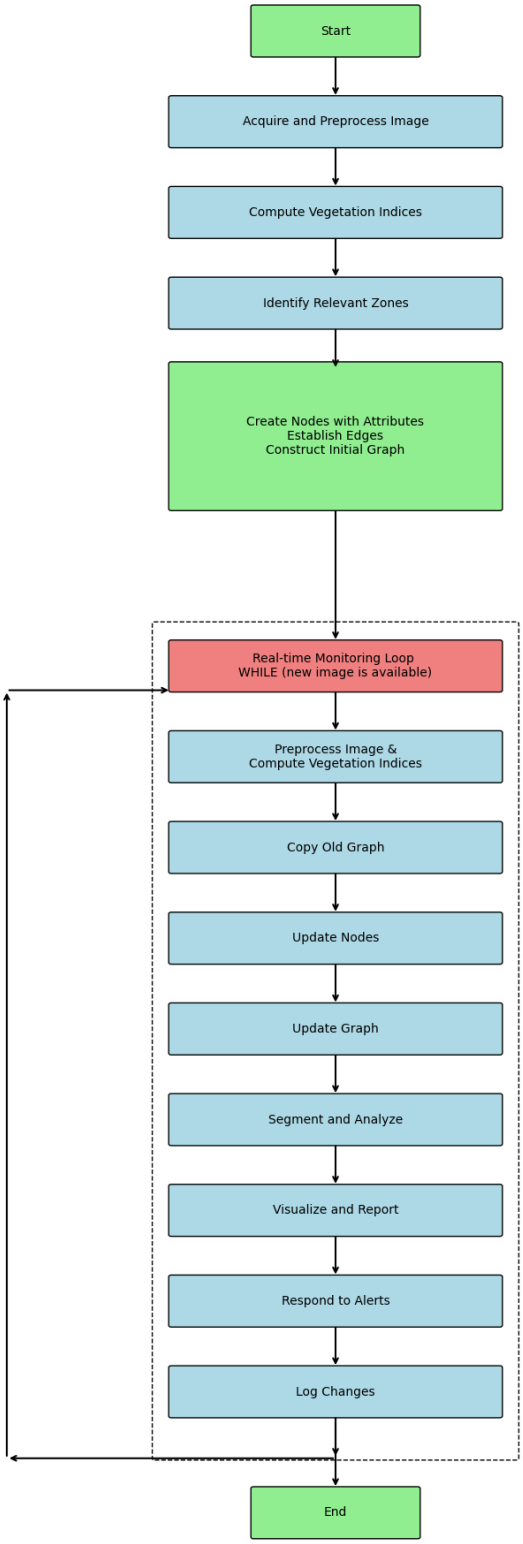

Figure 6.2: Dynamic Vegetation Monitoring Framework (DVMF)

Description of the vegetation indices

In order to provide further insight into the condition of vegetation and facilitate moni-

toring of this parameter over time, a number of vegetation indices were computed from the spectral data set for our exemplification. These indices simplify complex spectral data into interpretable metrics, allowing for the identification of specific attributes of vegetation, such as health, water content, and levels of stress. The following paragraphs presents the overview of the vegetation indices used in this study, accompanied by a discussion of their relevance and significance.

The Normalized Difference Vegetation Index (NDVI), the Normalized Difference Water Index (NDWI) and the Moisture Stress Index (MSI) are among the most important vegetation indices used in agricultural and environmental research studies. However, there are a series other indices that can be employed, depending on the specific circumstances.

The Normalised Difference Vegetation Index (NDVI), a widely used remote sensing vegetation index, quantifies vegetation greenness and serves as an important indicator of plant health and biomass. It it used in agricultural practices monitoring, environmental monitoring, and land cover classification. The NDVI measures plant health based on the differential reflection of red and near-infrared light, with healthy vegetation reflecting more NIR and less red light, resulting in higher NDVI values. NDVI is computed using the following formula:

$$NDVI = \frac{NIR - Red}{NIR + Red} \tag{6.1}$$

This index has values ranging between -1 and 1, with higher values, over 0.3, indicating areas covered with healthy green vegetation. Values close to zero are indicating areas with sparse or unhealthy vegetation, while values smaller than 0 typically correspond to areas without vegetation, such as water surfaces, barren soil, or building areas.

Another important vegetation index used in remote sensing is the Normalized Difference Water Index (NDWI). NDWI measures the water content in plants, and has values also from -1 to 1. Higher values indicate a greater water content. It is a useful tool for assessing the status of water within plants and for the detection of drought conditions. NDWI is computed using the following formula:

$$NDWI = \frac{NIR - SWIR}{NIR + SWIR} \tag{6.2}$$

While NDVI is a vegetation greenness index, NDWI is a water content index for vegetation. The two indices complement each other to provide a comprehensive assessment of vegetation health and water status.

The Moisture Stress Index (MSI) is also an indicator of plant water stress and it is computed using the following formula:

$$MSI = \frac{SWIR}{NIR} \tag{6.3}$$

Higher values indicate higher levels of moisture stress. For most healthy vegetation, MSI values can be less than 1.0, while values between 1.0 and 1.5 account for moderately stressed vegetation, and values greater than 1.5 usually indicating significant moisture stress.

Both the Normalized Difference Water Index (NDWI) and the MSI indices provide information about the water content and stress in vegetation. However, they do so from slightly different perspectives, and can complement each other effectively. The following section will explain why using both indices can be beneficial. The MSI is particularly sensitive to changes in the moisture content of vegetation and is used to detect water stress conditions. It facilitates the identification of regions where plants may be subjected to drought or an inadequate water supply. NDWI concentrates on the water content within vegetation canopies, taking into account the fact that plants can also suffer from an excess of water, a condition known as overwatering. This can be deleterious to plant health and can give rise to complications. Therefore, NDWI provides a measure of the quantity of water present in the vegetation, which can be essential for the assessment of overall plant health.

6.4 Conclusions

This chapter proves the significant potential of integrating dynamic networks with remote sensing data for advancing agricultural practices towards more sustainable and efficient methods. By using real-time data and targeted interventions, farmers can improve crop health, protect the environment, and reduce their carbon footprint, ultimately contributing to a more sustainable future. This integration supports not only agricultural productivity but also broader environmental concerns, ensuring that agricultural practices are in harmony with ecological sustainability.

Chapter 7

Conclusions

This thesis builds on the detailed objectives set out in the introduction by investigating existing algorithms and developing new ones. The main contributions are summarized below.

7.1 Research Contributions

The contributions of this research are diverse and address some gaps in the field of dynamic network algorithms.

1. The **Theoretical Background** and **Scoping Literature Review** chapters provided a detailed analysis of existing algorithms for SSSP and maximum flow problems, in static and dynamic situations, in terms of methodology and efficiency, identifying their strengths and limitations.

2. In **Dynamic Adjustment of Single Source Shortest Path** we developed an efficient algorithm that addresses the SSSP problem and focuses only on the affected parts of the network, ensuring optimal performance in dynamic scenarios.

3. In **Flow Increment through Network Expansion** we introduced an algorithm to solve the MCNEP, enhancing network flow capacities with minimal cost, applicable in various practical scenarios requiring dynamic flow augmentation.

4. In **Integration of Remote Sensing Data with Dynamic Network Processing** we proposed a method to combine remote sensing data with dynamic networks, offering a practical solution for real-time environmental and agricultural monitoring.

7.2 Limitations

Despite these contributions, the research has several limitations:

1. **Algorithm Validation**: While the dynamic SSSP algorithm is theoretically sound, it would require further validation through extensive real-world testing across diverse network scenarios.

2. **MCNEP generalization**: The proposed MCNEP algorithm needs to be expanded to better reflect real-world situations and costs of network changes. It should investigate some expansion scenarios where the cost functions are non-linear, such as polynomial or exponential.

3. **Integration Challenges**: The integration of remote sensing data with dynamic networks, though a promising avenue of research, is not without its challenges. At the time of writing, apart from the multispectral images, we did not have access to real and in-situ data. The incorporation of such data would be advantageous for the validation of use cases in these scenarios, as well as for the enhancement of the accuracy and applicability of the proposed solutions. Furthermore, the input of experts would be advantageous, as the specific considerations may vary depending on the particular situation, such as agricultural monitoring or environmental monitoring. Such expert insights could assist in addressing specific aspects that require consideration for more effective application.

7.3 Future works

In light of the findings and limitations of this research, several proposals for future work are presented.

1. **Enhanced Validation**: Extensive real-world testing of the proposed algorithms would be beneficial to ensure robustness and scalability across different network conditions and sizes.

2. **Algorithm Improvements**: Further refinement of the algorithms to handle more complex network dynamics and to improve computational efficiency. In particular, empirical testing for the delta parameter and investigation of the best data structures that can be used

in the dynamic SSSP algorithm could provide valuable insights into its optimal settings and performance.

3. **Generalisation of MCNEP**: An interesting direction for future research could be a generalisation of MCNEP.

4. **Parallelization**: It would be beneficial to investigate the potential for parallelisation of the dynamic algorithms, with a view to enhancing their computational efficiency and scalability. This could facilitate their suitability for real-time applications in large-scale networks.

5. **Broader Applications**: Expanding the integration of remote sensing data with dynamic networks to other fields, such as urban planning and disaster management, where real-time data can significantly impact decision-making processes.

In conclusion, this thesis has made some contribution to the advancement of the understanding and development of dynamic algorithms for network problems. While there are challenges and limitations, the proposed solutions offer a solid foundation for future research and practical applications, contributing to the ongoing evolution of network algorithms and their integration with real-world data.

Bibliography

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms and applications*. Prentice hall, 1995.

[2] Avishai Ceder. *Public transit planning and operation: Modeling, practice and behavior*. CRC press, 2016.

[3] Laura Ciupala, Adrian Deaconu, and Luciana Majercsik. "Shortest paths in a diagraph with an underestimated arc weight". In: *Bulletin of the Transilvania University of Brasov. Series III: Mathematics and Computer Science* (2022), pp. 193–196.

[4] Adrian Deaconu. "The inverse maximum flow problem considering l∞ norm". In: *RAIRO-Operations Research* 42.3 (2008), pp. 401–414.

[5] Adrian Deaconu and Eleonor Ciurea. "The inverse maximum flow problem under norms". In: *Carpathian Journal of Mathematics* (2012), pp. 59–66.

[6] Adrian Marius Deaconu and Luciana Majercsik. "Flow Increment through Network Expansion". In: *Mathematics* 9.18 (2021), p. 2308.

[7] Amir Elalouf, Ron Adany, and Avishai Avi Ceder. "Flow expansion on transportation networks with budget constraints". In: *Procedia-Social and Behavioral Sciences* 54 (2012), pp. 1168–1175.

[8] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. "Fully dynamic algorithms for maintaining shortest paths trees". In: *Journal of Algorithms* 34.2 (2000), pp. 251–281.

[9] T Karpagam et al. "Flow Based Algorithm". In: *American Journal of Applied Sciences* 9.2 (2012), p. 238.

[10] Myint Than Kyi and Lin Lin Naing. "Application of Ford-Fulkerson algorithm to maximum flow in water distribution pipeline network". In: *International Journal of Scientific and Research Publications* 8.12 (2018), pp. 306–310.

[11] Ulrich Meyer and Peter Sanders. "Δ-stepping: a parallelizable shortest path algorithm". In: *Journal of Algorithms* 49.1 (2003), pp. 114–152.

[12] Paolo Narvaez, Kai-Yeung Siu, and Hong-Yi Tzeng. "New dynamic algorithms for shortest path tree computation". In: *IEEE/ACM Transactions On Networking* 8.6 (2000), pp. 734–746.

[13] Paolo Narvaez, Kai-Yeung Siu, and Hong-Yi Tzeng. "New dynamic SPT algorithm based on a ball-and-string model". In: *IEEE/ACM transactions on networking* 9.6 (2001), pp. 706–718.

[14] James Orlin. "A faster strongly polynomial minimum cost flow algorithm". In: *Proceedings of the Twentieth annual ACM symposium on Theory of Computing*. 1988, pp. 377–387.

[15] James B. Orlin. "A faster strongly polynomial time algorithm for submodular function minimization". In: *Mathematical Programming* 118.2 (2013), pp. 237–251. DOI: 10.1007/s10107-006-0079-7.

[16] Ioana Cristina Plajer, Alexandra Baicoianu, and Luciana Majercsik. "AI-based visualization of remotely-sensed spectral images". In: *2023 International Symposium on Signals, Circuits and Systems (ISSCS)*. IEEE. 2023, pp. 1–4.

[17] Ioana Cristina Plajer et al. "NDVI Computation from Hyperspectral Images". In: *2023 13th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. IEEE. 2023, pp. 1–5.

[18] Ioana Cristina Plajer et al. "Multisource Remote Sensing Data Visualization Using Machine Learning". In: *IEEE Transactions on Geoscience and Remote Sensing* (2024).

[19] Ganesan Ramalingam and Thomas Reps. "An incremental algorithm for a generalization of the shortest-path problem". In: *Journal of Algorithms* 21.2 (1996), pp. 267–305.

[20] Alexander Schrijver. "On the history of the transportation and maximum flow problems". In: *Mathematical programming* 91 (2002), pp. 437–445.

[21] Javad Tayyebi and Adrian Deaconu. "Inverse generalized maximum flow problems". In: *Mathematics* 7.10 (2019), p. 899.

[22] Javad Tayyebi, Abumoslem Mohammadi, and Seyyed Mohammad Reza Kazemi. "Reverse maximum flow problem under the weighted Chebyshev distance". In: *RAIRO-Operations Research-Recherche Opérationnelle* 52.4-5 (2018), pp. 1107–1121.

[23] Claudemir Duca Vasconcelos et al. “Network flows modeling applied to the natural gas pipeline in Brazil”. In: *Journal of natural gas science and engineering* 14 (2013), pp. 211–224.

[24] Zhao Zhang and Xiaohui Huang. “Discrete Newton Method”. In: *Nonlinear Combinatorial Optimization*. Ed. by Ding-Zhu Du, Panos M. Pardalos, and Zhao Zhang. Cham: Springer International Publishing, 2019, pp. 37–56. ISBN: 978-3-030-16194-1. DOI: 10.1007/978-3-030-16194-1_2. URL: https://doi.org/10.1007/978-3-030-16194-1_2.

[25] Amanda Ziemann. “A manifold learning approach to target detection in high-resolution hyperspectral imagery”. PhD thesis. Apr. 2015. DOI: 10.13140/RG.2.1.2503.3680.