ROUTING ALGORITHMS AS AN APPLICATION OF GRAPH THEORY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÖKBERK YILDIRIM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

JANUARY 2023

Approval of the thesis:

ROUTING ALGORITHMS AS AN APPLICATION OF GRAPH THEORY

submitted by **GÖKBERK YILDIRIM** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Selçuk Kestel _____
Director, Graduate School of **Applied Mathematics**

Assoc. Prof. Dr. Oğuz Yayla _____
Head of Department, **Cryptography**

Prof. Dr. Ferruh Özbudak _____
Supervisor, **Cryptography, METU**

Examining Committee Members:

Prof. Dr. Ferruh Özbudak
Institute of Applied Mathematics, METU _____

Assoc. Prof. Dr. Oğuz Yayla
Institute of Applied Mathematics, METU _____

Assoc. Prof. Dr. Burcu Gülmez Temur
Department of Mathematics, Atılım University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: GÖKBERK YILDIRIM

Signature :

ABSTRACT

ROUTING ALGORITHMS AS AN APPLICATION OF GRAPH THEORY

Yıldırım, Gökberk
M.S., Department of Cryptography
Supervisor : Prof. Dr. Ferruh Özbudak

January 2023, 45 pages

This paper examines routing algorithms that are generally used in various network types, such as Internet Protocol in graph-based models, to find the shortest routing path or minimum cost. The fundamental approach for calculating the shortest path between one point to another is searching a given graph, starting at the source node, and traversing adjacent nodes until the destination node is reached. The aim is to identify the shortest routing path to the destination node. This paper searches well-known routing algorithms, namely Bellman-Ford and Dijkstra's single source shortest path algorithms, and their application areas like network communication protocols, a cryptocurrency exchange in arbitrage, and vertex-weighted directed graphs in robotics. The paper aims to decrease the number of blocked path requests and improve overall usage, especially in cryptographic tools. The routing algorithms are analyzed and defined with uncertainty arising from various factors, such as weather conditions and road capacity at specific times. The main challenges in the Shortest Path Algorithms (SPP) are identifying which edges to add and comparing the distances between different paths based on their edge lengths.

Keywords: Bellman-Ford, Dijkstra, shortest path, routing algorithms, communication networks, directed graphs, minimum cycle path

ÖZ

GRAFİK TEORİSİNİN BİR UYGULAMASI OLARAK YÖNLENDİRME ALGORİTMALARI

Yıldırım, Gökberk
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Prof. Dr. Ferruh Özbudak

Ocak 2023, 45 sayfa

Bu makale, en kısa yönlendirme yolunu veya minimum maliyeti bulmak için grafik tabanlı modellerde İnternet Protokolü gibi genellikle çeşitli ağ türlerinde kullanılan yönlendirme algoritmalarını incelemektedir. Bir noktadan diğerine en kısa yolu hesaplamak için temel yaklaşım, kaynak düğümden başlayarak belirli bir grafiği aramak ve hedef düğüme ulaşılana kadar bitişik düğümleri katetmek. Amaç, hedef düğüme giden en kısa yönlendirme yolunu belirlemektir. Bu makale, iyi bilinen yönlendirme algoritmalarını, yani Bellman-Ford ve Dijkstra'nın tek kaynaklı en kısa yol algoritmalarını ve ağ iletişim protokolleri, arbitrajda bir kripto para birimi değişimi ve robotikte köşe ağırlıklı yönlendirilmiş grafikler gibi uygulama alanlarını araştırmaktadır. Rapor, engellenen yol isteklerinin sayısını azaltmayı ve özellikle kriptografik araçlarda genel kullanımı iyileştirmeyi amaçlıyor. Yönlendirme algoritmaları, belirli zamanlarda hava koşulları ve yol kapasitesi gibi çeşitli faktörlerden kaynaklanan belirsizliklerle analiz edilir ve tanımlanır. En Kısa Yol Algoritmalarındaki (SPP) ana zorluklar, hangi kenarların ekleneceğini belirlemek ve farklı yollar arasındaki mesafeleri kenar uzunluklarına göre karşılaştırmaktır.

Anahtar Kelimeler: Bellman-Ford, Dijkstra, en kısa yol, yönlendirme algoritmaları, iletişim ağları, yönlendirilmiş grafikler, minimum döngü yolu

ACKNOWLEDGMENTS

I would like to express my great appreciation to my thesis supervisor Prof. Dr. Ferruh Özbudak, for his patient guidance, thrust, enthusiastic encouragement, and valuable advice during the development and preparation of this thesis. His willingness to give his time and share his experiences has brightened my path. Thanks to his support, completing this thesis based on my career plan is a significant chance. Additionally, my great appreciation is for my examining committee members for their patience, interest, and helpfulness.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

i	Source Node
j	Destination Node
k	Intermediate Node
N	List of nodes in a network
S	A permanent set of nodes that have been considered so far in the calculation for Dijkstra's algorithm
S'	Tentative list of nodes in Dijkstra's algorithm (that are yet to be considered in the calculation)
N_k	List of neighboring nodes of node k
d_{ij}	Cost of the link between node i and node j
$d_{ij}(t)$	Link cost between nodes i and j at time t
$\overline{D}_{ij}$	Minimum cost of the path computed from node i to node j for Bellman–Ford
$\overline{D}_{ij}^{(h)}$	Minimum cost of the path computed from node i to node j when h hops have been considered
$\overline{D}_{ij}(t)$	Minimum cost of the path computed from node i to node j at time t
$d_{kj}^{i}(t)$	Link cost between nodes k and j at time t as known to node i
$\overline{D}_{kj}^{i}(t)$	Minimum cost of the path computed from node k to node j at time t as known to node i
$\underline{D}_{ij}$	Minimum cost of the path computed from node i to node j (Dijkstra)

CHAPTER 1

INTRODUCTION

Graph theory is a mathematical discipline that studies the properties of graphs, which are used to model a variety of different systems in science and engineering. In the field of computer networks, graph theory is used to represent and analyze the connectivity of a network. Routing algorithms are a crucial application of graph theory in networking, as they use graph-based models to determine the best path for transmitting data between devices in a network. A graph is constructed by vertices (or nodes) that are connected by edges. The edges of a graph can be directed (e.g., one-way streets in a city map) or undirected (e.g., two-way streets in a city map). The vertices and edges of a graph can also have weights associated with them, which represent the cost or distance of traversing the edge.

Routing algorithms use graph-based models to represent the connectivity of a network and determine the shortest path between two nodes. Many routing algorithms, such as Dijkstra's algorithm and Bellman-Ford's algorithm, are based on graph theory principles and use techniques from graph theory to find the shortest path. The Bellman-Ford algorithm, first proposed by Bellman in 1958, is a popular routing algorithm that has been widely adopted in both network routing [17]. The Dijkstra algorithm, first proposed by Dijkstra in 1959, is another widely-used routing algorithm that has been shown to be effective in a variety of settings, both in network routing and cryptography, see [4]. In addition to finding the shortest path between two nodes, routing algorithms also need to consider other factors, such as network congestion, link reliability, and security. These factors can be incorporated into the graph-based model used by the routing algorithm and used to determine the most optimal path.

For example, in a congested network, a routing algorithm may choose a path with fewer hops (i.e., fewer edges in the graph) even if it has a higher cost or distance, as this path may be less congested and provide better performance. Similarly, in a network with unreliable links, a routing algorithm may choose a path with more reliable connections even if it has a higher cost or distance.

Security is another important factor that needs to be considered in routing algorithms. Cryptography can be used to secure the communication of routing information between devices and prevent spoofing attacks, in which an attacker impersonates a legitimate device in order to gain access to the network or manipulate routing information. According to Jones [9] in 2020, routing algorithms play a crucial role in modern cryptography, as they help to secure the transmission of data across networks. Thus, both the Bellman-Ford and Dijkstra algorithms can be enhanced with the use of cryptography to ensure the confidentiality and integrity of routing information.

In summary, routing algorithms are a vital application of graph theory in networking, as they use graph-based models to determine the best path for transmitting data between devices in a network. Routing algorithms need to consider various factors, such as network congestion and link reliability, in order to find the most optimal path. In addition, the use of cryptography in routing algorithms helps to protect the confidentiality and integrity of routing information and ensure the secure operation of the network.

1.1 Motivation and Problem Definition

Shortest path algorithms in computer networks are widely used in many areas, such as communication networks, social networks, and genetics. In this thesis, shortest-path algorithms will be applied to communication networks to find the optimal paths between source and destination nodes by minimizing costs. In this chapter, we will prepare a base for the shortest-path algorithms since we will focus on the communication network and other technical terms because when it comes to building routing protocols, shortest-path algorithms are ubiquitous. Even though there are different classes of routing algorithms to solve different kinds of problems, only the shortest

path routing will be the leading interest for our purpose.

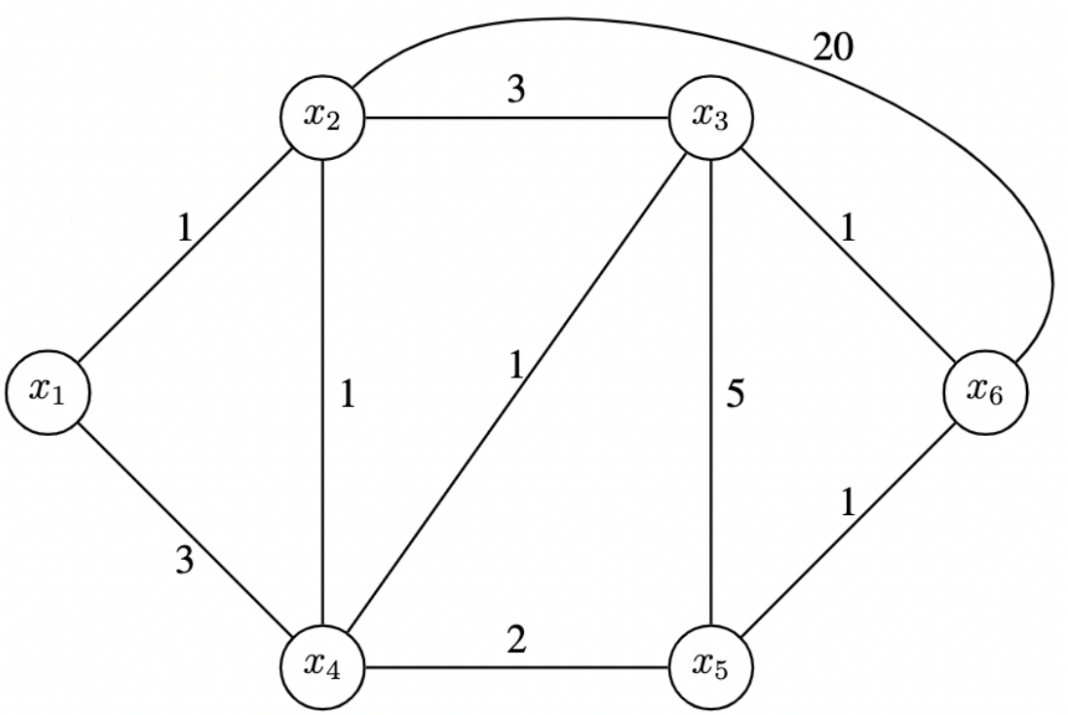

Figure 1.1: A six node network

In most cases, the building blocks of a communication network are nodes and links. The names of the network and nodes may vary due to the usage of different network types. For the Internet Protocol (IP) network, a node is denoted by a router, whereas the same node is called the central office or toll switch in the telephone network [21]. An optical or electro-optical switch is a link node in a network that uses light as its medium. IP trunk or IP link refer to a connection between two sites, in this case, two routers in an IP network. The endpoint of this connection as it exits a router is known as an interface. A communication network allows for smooth data flow between the origin and destination nodes. In a nutshell, we call the node where traffic first appears the source node and the one where it finally arrives at the destination node. Look at the diagram of a network with six nodes shown in Figure 1.1. In this network, it is supposed to be that we have traffic flow that starts from node x_1 to node x_2. This flow is not only the flow that can be defined. For instance, another traffic can be described from node x_2 to node x_5; in this case, the source node would be node x_2, and the destination node would be node x_5. These are the two sample flows. Apart from these, there can be derivatized much more traffic between nodes.

The direct traffic must flow from the source node to the destination node in a communication network. Additionally, a path or route defined between nodes can be constructed manually and called a static route. On the contrary, routing algorithms are being used to determine a route quickly and conveniently because routing algorithms take care of any communication network's requirements and any additional objectives that a service provider wishes to set for itself.

In general, communication networks have different types because objectives may differ. These types can be examined under two broad categories such as network-focused and user-focused. That is, the user-focused network has to deliver outstanding service for the clients or end users to let traffic flow from the source node to the destination node quickly for each user. On the other hand, this means that the actions taken should not negatively impact the experience of other users or disrupt connections between other source and destination nodes within the communication network. In network-focused communication networks, most users have efficient and equitable routing, and the provisioned services undoubtedly provide exceptional routing functionalities. This type of network is essential because the resources, such as capacity, are limited [21].

Next, we consider two essential algorithms that profoundly impact data networks, especially routing on the Internet. These algorithms are famously referred to as the Bellman-Ford algorithm and Dijkstra's algorithm. They can be categorized as user-focused, as per the broader categories discussed earlier. The thesis describes two routing algorithms that are called shortest path algorithms. These algorithms aim to find the most efficient path between two nodes, the source and the destination. The concept of the shortest path can be understood by using the example of a road network, where the shortest path is the one with the least distance. However, the concept of the shortest path can also be applied in other ways, such as finding the quickest route between two points in terms of time. The distance between nodes does not need to be physical; it can be measured in different ways, such as coefficients, scores, meters, and time. The primary consideration is that these efficient algorithms should be able to measure units by generically applying them to a given network because algorithms are responsible for handling each link between nodes.

The term 'distance cost', 'link metric', 'cost', or 'link cost' is a generic terms used in communication networks to guide a measuring distance without specifying its units. For example, in Figure 1.1, a value is assigned to each link, such as link $x_2 - x_6$ having the value of 20, and this value is referred to as the link cost, distance cost, or link metric of link $x_2 - x_6$. No specific measurement rule is taken into account for computing the units in the network, and the popularly used ones are miles, kilometers, or minutes. Upon examination, it is readily apparent that the optimal path between nodes x_1 and x_6 is the sequence of $x_1 - x_2 - x_4 - x_3 - x_6$, with a minimum cost of 4. Notably, this path does not include the link between x_2 and x_6, although, from the perspective of the number of visited nodes, it may seem as though the path of $x_1 - x_2 - x_6$ is the shortest. This would apply if the cost of the link was determined by the number of nodes visited or the number of intermediate connections. To elaborate, if the number of intermediate connections is a crucial factor in determining distance within a particular network, then the network in Figure 1.1 can be evaluated by assigning a cost of one to each link instead of the number indicated in the figure. An algorithm that can operate without relying on the specific cost assigned to each link is advantageous; this is where the Bellman-Ford algorithm and Dijkstra's algorithm come in handy. There is another approach to calculating the shortest path, thanks to the linear programming approach.

In the computation of the shortest path, it is common to utilize the additive property to determine the overall distance of a path by summing the cost of each link along the path. Therefore, we will begin by assuming this property for shortest path routing when discussing the Bellman-Ford algorithm and Dijkstra's algorithm and their variations. To identify the most optimal path, one can determine the distance cost between two nodes using non-additive concave properties. To put it simply, algorithms that employ a non-additive concave property are commonly referred to as the widest path routing algorithms [21]. However, in this thesis, we will not be exploring this type of algorithm.

To end this section, it is crucial to understand the connection between a network and a graph. Due to the close relationship between graph and network, networks can be thought of as a graph by connecting every node to a particular edge with links. Links mean an edge connecting nodes or vertices to each other in a network and can have

more weights, such as bandwidth, convenience coefficient, delay, and cost. As an example, Figure 1.1 illustrates a network graph that contains six nodes and ten links. The links in the figure are also assigned to costs or weight.

1.2 The Outline of the Thesis

The structure of the thesis work is as follows:

- **Chapter 2** covers the necessary background information by providing definitions and fundamental concepts of graph theory, with a focus on understanding the characteristics of communication networks that are relevant to the thesis.
- **Chapter 3** relates the Bellman-Ford algorithm and the distance vector approach to compute the shortest path between node i to node j in terms of two different views namely centralized and distributed.
- **Chapter 4** describes Dijkstra's algorithm for finding the shortest path from the source node to the destination node by evaluating both centralized and distributed approaches.
- **Chapter 5** analyzes the differences between Bellman-Ford and Dijkstra's algorithms based on their nature, performance, and complexity.
- **Chapter 6** examines other application areas of routing algorithms that are used in the thesis such as vertex-weighted directed graphs and minimum cycle detection (or hamiltonian path) for arbitrage in a directed graph. Additionally, it investigates the complexities of these two algorithms on different graph structures.
- **Chapter 7** concludes the paper thanks to comprehensive information about existing routing algorithms and gives a brief overview of possible future work.

CHAPTER 2

PRELIMINARY TO THE SUBJECT

2.1 Notions of Graphs

This paragraph explains that the definition of a graph can vary depending on the field of study, but it will provide a common definition to clarify the concepts of graph theory. It also notes that for any two elements, u and v, in a set V, the unordered pair $\langle u, v\rangle$ is used to represent the relationship between them. Pair of u and v need not be distinct and the order in which they appear in the unordered pair does not matter. Unordered pairs $\langle u, v\rangle$ where $u, v \in V$ is defined by $(V \times V)^{'}$.

Definition 2.1.1. *A graph (network), denoted by G, can be defined as a collection of vertices (nodes), represented by the set V, and a set of edges, represented by E that is $G = (V, E)$, such that the edges are a subset of all possible connections between the vertices, represented by $(V \times V)^{'}$. The number of vertices in the graph G is known as the order of the graph, and it is often referred to as a graph on the set V.*

Barnes [3] in 1969 highlighted that the order of a graph can be infinite which means the graph has infinitely many nodes or vertices. In practice, we will primarily consider finite graphs, which have a limited number of vertices. Infinite graphs can have theoretical significance, but we will concentrate on networks that exist in the real world and they have a finite number of vertices. It is important to note that unless otherwise stated, all the graphs we discuss will be assumed to be finite.

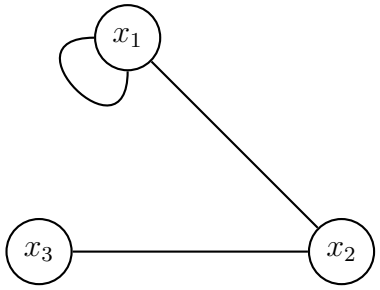

Figure 2.1: A sample undirected graph representation

Example 2.1.2. *Consider a set of vertices $V = \{x_1, x_2, x_3\}$. The set of all possible connections between these vertices, represented by $V \times V$, and E should be a subset of this set, where the connections are symmetric. For example, if we take $E = \{\langle x_1, x_1\rangle, \langle x_1, x_2\rangle, \langle x_2, x_3\rangle\}$, To depict the graph, we label and identify individual vertices, and draw lines to connect them if an edge exists, as illustrated below:*

Definition 2.1.3. *Let G be a graph composed of vertices V and the set of edges E, represented by $G = (V, E)$. A connection of the form $\langle v, v\rangle \in E$ is referred to as a loop. If the graph G does not contain any loops, it is known as a simple graph.*

We can create a simple graph from any given graph by removing all loops. For example, the previous graph can be transformed into a simple graph by eliminating all loops present in it.

$$E = \{\langle 1, 2\rangle, \langle 2, 3\rangle\}$$

Note that we will assume all the graphs are simple unless otherwise specified.

An unordered pair $\langle u, v\rangle$ is nothing but a simple set $\{u, v\}$, which is a subset of V of size 2. Except for the simple graphs, others might have loops connecting a node itself, and it is defined as $\{v, v\} = \{v\}$ with size 1. Therefore, the choice of angle brackets can be replaced with curly braces in order to represent a set of edges like below:

$$E = \{\{1, 2\}, \{2, 3\}\}$$

Many graphs that we encounter in real-world situations are not naturally symmetric,

or undirected, as seen in the works of [6]. For example, when constructing a graph of webpages, a directed link is drawn from one webpage to another if there is a hyperlink present, and in relationship graphs, a directed link is created from relation X to relation Y if relation X has a friendship with relation Y. These types of graphs necessitate us to consider links as ordered pairs (u, v), as opposed to unordered pairs u, v, resulting in a specific definition for them [20].

Definition 2.1.4. *A graph is said to be directed, denoted by $G = (V, E)$ if it consists of vertices V and edges E where the edges are a subset of $V \times V$. The edges in E are referred to as directed edges or links. If the graph G does not have any loops, we refer to it as a simple directed graph.*

Suppose an edge e is directed edge and connecting u and v starting at u and ending at v, denoted by $e = (u, v) \in E$. u is the initial vertex of e, and v is the final vertex of e.

Example 2.1.5. *Let $V = \{x_1, x_2, x_3\}$ and $E = \{\langle x_1, x_3\rangle, \langle x_3, x_1\rangle, \langle x_3, x_2\rangle\}$, we can use pair (u, v) in E if $(v, u) \in E$ for representation of the directed graph and an edge will be drawn from node u to node v with an arrow pointing towards v. If both (u, v) and (v, u) are in E, nodes can be represented as two distinct edges with arrows at both ends.*

Figure 2.2: A sample directed graph in two different representations

Due to the fact that edges in directed graphs are conceived of as having direction, this digraph contains three edges rather than two, despite what the illustration on the left might lead one to expect. Note that we can convert an undirected graph into a directed graph by replacing an edge between nodes with two directed edges in opposite directions. For instance, if $G = (V, E)$ is undirected graph, an directed graph $G' = (V, E')$ can be constructed with the same set of vertices V, but with an edge set E' that includes two directed edges for every edge in E, one in each direction.

$$E' = \{(u, v), (v, u) : \langle u, v \rangle \in E\} \tag{2.1}$$

To transform undirected graph G into a directed graph G', (u, v) and (v, u) edges must include for any edge $\langle u, v \rangle \in G$ as in the Eq. 2.1. In the case of non-simple graphs, there is a loop $\langle v, v \rangle$ that can also be transformed into (v, v). For instance, the set of edge looks like $E' = \{(x_1, x_1), (x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2)\}$. The directed graph equivalent to the undirected graph can then be illustrated in the following manner.

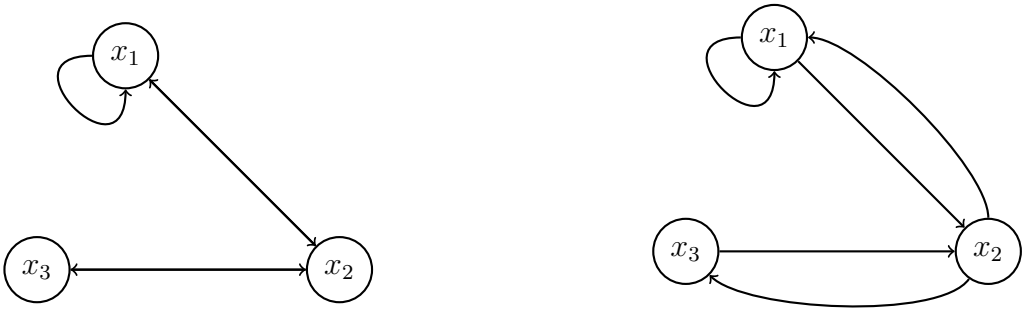

Figure 2.3: A sample directed graph with loops in two different representations

In simpler terms, undirected graphs can be viewed as a specific type of directed graph where the edges in the set E are symmetric, meaning that if an edge (u, v) is present in the set E, then the edge (v, u) is also present. This is due to the fact that in undirected graphs, edges are represented as unordered pairs of elements of set V, which is a symmetric subset of the set $V \times V$ of ordered pairs hence the notation $(V \times V)'$. The main difference between undirected and directed graphs is that in counting the edges, the directed graph will have twice as many edges as the undirected graph when considering simple graphs as an example. This perspective is useful as it allows us to examine both directed and undirected graphs using a consistent approach. As a result, it is common to use the notation of ordered pairs (u, v) for edges in an undirected graph in this context. Additionally, the edge set can be represented as a symmetric subset of $V \times V$, meaning that $(u, v) \in E$ and $(v, u) \in E$ are equivalent.

CHAPTER 3

BELLMAN-FORD ALGORITHM AND THE DISTANCE VECTOR APPROACH

The Bellman-Ford algorithm is a method for finding the optimum or shortest path between a source node and a destination node in a network. According to a study by Smith [5] in 2010, the Bellman-Ford algorithm is highly effective at solving the shortest path problem with a single source. In a distributed setting, a distance vector approach represents the shortest paths. It is highly recommended to see this book [8] in order to obtain more detailed information about the Bellman-Ford algorithm theoretically. In this chapter, we Bellman-Ford's centralized and distributed approaches will be discussed.

3.1 CENTRALIZED VIEW OF BELLMAN-FORD ALGORITHM

Two specific nodes, labeled i and j, will be utilized in a network including N nodes for the centralized Bellman-Ford algorithm. These nodes may be directly linked through connections x_1-x_6 with destination nodes x_2 and x_6 as shown in Figure 1.1.

The Bellman-Ford algorithm is used to find the shortest path between two nodes in a network, even if they are not directly connected. In Figure 1.1, for instance, nodes x_1 and x_6 are not directly connected. This leads to an important concept. Hence, regardless of whether nodes are connected directly, nodes should be considered with the cost of the links. We introduce two necessary notations for this concept:

d_{ij} := Cost of the link between node i and node j

$\overline{D_{ij}}$:= Minimum cost of the path computed from node i to node j

Various notations are used to distinguish between calculations of different types of algorithms, such as hats, over-bars, and underscores. For example, in the Bellman-Ford algorithm, overbars are used for distance computations and variations. All the notations used in all the chapters are explained on the abbreviations page.

When two nodes have a direct connection, the cost of the link d_{ij} between them is considered to be a finite number. In Figure 1.1, node x_2 and node x_6 are connected directly with a link cost of 20. That is, link cost of node x_2 and node x_6 can be written $d_{x_2x_6}$ = 20. However, node x_1 and node x_6 are not connected directly, so the link cost is $d_{x_1x_6}$ = ∞. Now, the difference between d_{ij} and $\overline{D}_{ij}$ will be considered. From the nodes x_2 and x_6, it can be seen that the minimum cost is essentially 3, and the path for that cost is x_2-x_4-x_3-x_6; that is, $\overline{D}_{x_2x_6}$ = 3 while $d_{x_2x_6}$ = 20. For nodes x_1 and x_6, we find that $\overline{D}_{x_1x_6}$ = 5 while $d_{x_1x_6}$ = ∞. This result implies that two nodes in a network can have a minimum path cost even when they are not directly connected. That means it is impossible to say that one of the end nodes is totally different and isolated from the other nodes in the network.

The shortest-path algorithms are the leading actor in calculating the minimum cost between two nodes. From the six-node network example, intermediary nodes are essential, and they should be considered for finding the shortest path in a given network to understand such an algorithm. Suppose that node k is directly connected to one of the destination nodes in the network; the distance between node k and j has a finite number represented as d_{kj}. For this reason, any inquiry regarding Bellman-Ford's equations must be answered by the shortest path between node i and node j.

$$\begin{aligned} &\overline{D}_{ii} = 0, \quad \text{for all i} \\ &\overline{D}_{ij} = \min_{k \neq j}\{\overline{D}_{ik} + d_{kj}\}, \quad \text{for i} \neq \text{j} \end{aligned} \tag{3.1}$$

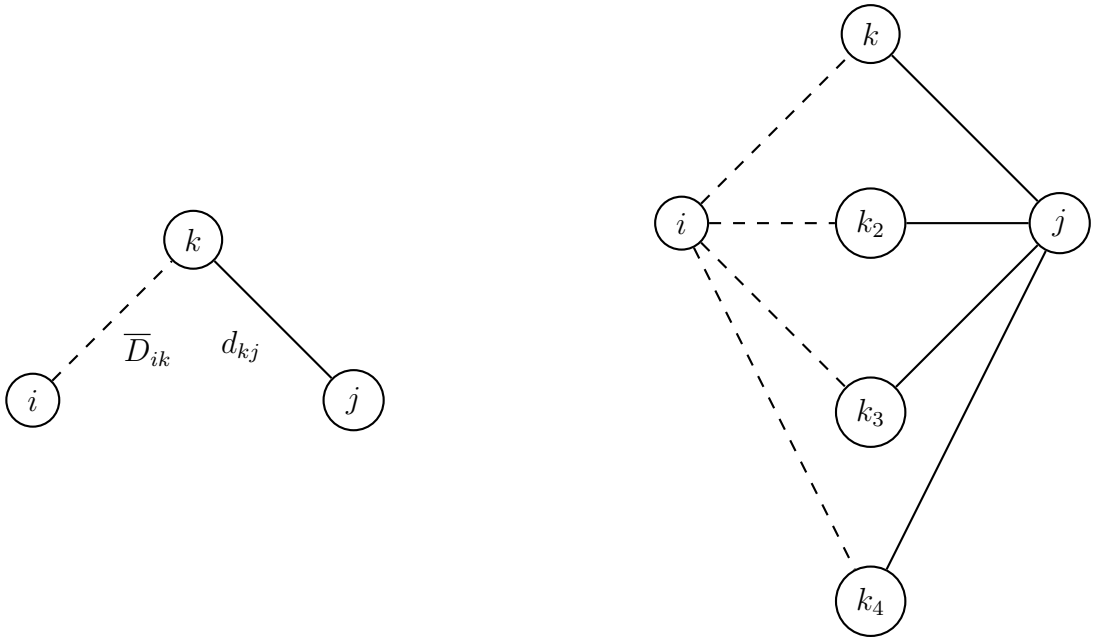

Figure 3.1: Centralized Bellman–Ford Algorithm with direct links in solid lines and distances in dashed lines.

Eq. 3.1 emphasizes that for a pair of nodes i and j. It also explains the minimum cost from node i to intermediate node k and their distance link $k - j$ cost d_{kj} where those are directly connected. All the cost computation is represented in Figure 3.1. It is possible to see that multiple intermediary nodes k might be directly tied to the destination node j marked as k, k_2, k_3, and k_4 in Figure 3.1 where $k = i$ is concerned. Thus, all the ks falling into this category have minimum costs and can be computed. Additionally, a node k, which is not connected to j directly, has distance d_{kj} is equal to ∞. For such, ks must be considered as a part of the solution. As a result, the minimum cost of the path computation remains the same. According to Eq. 3.1, from node i to intermediary node k has the minimum cost, which is $\overline{D}_{ik}$. As a result, the minimum cost after repeating the process for the number of hops h thanks to a variation of Eq. 3.1 needs to be defined as follows:

$\overline{D}_{ij}^{(h)}$:= Minimum cost of the path computed from node i to node j when h hops have been considered

The Bellman-Ford algorithm is an iterative process that iterates the number of hops given in Algorithm 1 taken from the book [21].

Algorithm 1 Centralized Bellman-Ford Algorithm

for nodes i and j in network do

$$\overline{D}_{ii}^{(0)} = 0, \quad \text{for all i} \quad \overline{D}_{ij}^{(0)} = \infty, \quad \text{for i} \neq \text{j} \tag{3.6}$$

end for

for (h = 0 to N - 1) do

$$\overline{D}_{ii}^{(h+1)} = 0, \quad \text{for all i} \tag{3.7}$$

$$\overline{D}_{ij}^{(h+1)} = \min_{k \neq j} \{\overline{D}_{ik}^{(h)} + d_{kj}\}, \quad \text{for i} \neq \text{j} \tag{3.8}$$

end for

Table 3.1: Minimum cost of node x_1 to other nodes using Algorithm 1

h	$\overline{D}_{x_1x_2}^{(h)}$	Path	$\overline{D}_{x_1x_3}^{(h)}$	Path	$\overline{D}_{x_1x_4}^{(h)}$	Path	$\overline{D}_{x_1x_5}^{(h)}$	Path	$\overline{D}_{x_1x_6}^{(h)}$	Path
0	∞	-	∞	-	∞	-	∞	-	∞	-
1	1	x_1-x_2	∞	-	3	x_1-x_4	∞	-	∞	-
2	1	x_1-x_2	4	x_1-x_2-x_3	2	x_1-x_2-x_4	5	x_1-x_4-x_5	21	x_1-x_2-x_6
3	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	5	x_1-x_2-x_3-x_6
4	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	4	x_1-x_2-x_4-x_3-x_6
5	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	4	x_1-x_2-x_4-x_3-x_6

The Bellman-Ford algorithm is applied to the six-node network as visualized in Figure 1.1 to find all the minimum costs from node x_1 to all the other nodes and exemplified in Table 3.1. An excellent way to understand the basis of the hop-based iteration of the Bellman-Ford approach is to see an example. If we calculate the shortest path between node x_1 and node x_6 while the number of hops increases, computed shortest paths will change. Consider $h = 1$; the minimum cost is $D_{x_1x_6}^{(1)} = \infty$ due to the fact that there is no direct link between x_1 and x_6. For $h = 2$, the only possible path seems to be $x_1 - x_2 - x_6$, which is two hopped link path. In other words, it uses two connections, one between x_1 and x_2, and another between x_2 and x_6, resulting in a minimum cost of 21, as represented by $\overline{D}_{x_1x_6}^{(2)}$. At $h = 3$, there are two possible paths to reach node x_6 where $d_{kx_6} < \infty$ for only k that are shown below:

$$k = 3, \quad \overline{D}^{(2)}_{x_1 x_3} + d_{x_3 x_6} = 4 + 1 = 5$$
$$k = 5, \quad \overline{D}^{(2)}_{x_1 x_5} + d_{x_5 x_6} = 5 + 1 = 6$$

Finally, for $h = 4$, Bellman-Ford step as follows where $d_{kx_6} < \infty$:

$$k = 3, \quad \overline{D}^{(3)}_{x_1 x_3} + d_{x_3 x_6} = 3 + 1 = 4$$
$$k = 5, \quad \overline{D}^{(3)}_{x_1 x_5} + d_{x_5 x_6} = 4 + 1 = 5$$
$$k = 2, \quad \overline{D}^{(3)}_{x_1 x_2} + d_{x_2 x_6} = 1 + 20 = 21$$

For this scenario, we select the first one since the minimum cost from node x_1 to node x_6 is 4, i.e., $\overline{D}^{(4)}_{x_1 x_6} = 4$ with the shortest path $x_1 - x_2 - x_4 - x_3 - x_6$. Note that the computation of the minimum cost is the only thing the Bellman-Ford algorithm can do; it does not keep a record of the most efficient route, but one can refer to Table 3.1 to understand the algorithm's operation as it displays the most direct route. In most networking environments, knowing the entire path is optional. The thing is that the next node k with minimum cost suffices. The next node can be readily identified using the Eq 3.8.

Figure 3.2: Distance vector view for calculating the shortest path

3.2 DISTRIBUTED VIEW OF A DISTANCE VECTOR APPROACH

For real-life situations, the nodes in the distributed view must determine the shortest path between nodes, especially to a destination node. According to the centralized view mentioned above, before reaching the destination node, the source node must

have knowledge about the cost of the shortest path to other nodes. That means the minimum cost of the destination $\overline{D}_{ik}^{(h)}$ should be calculated by using Eq 3.8 where ik communicates through Figure 3.1. There needs to be more than this approach for the centralized view of the Bellman-Ford algorithm to be used in a distributed view. Therefore, some minor modifications to minimize the cost of the computation should be initiated. Consider the change in Eq 3.7 and use this for the following step to find the minimum distance as follows:

$$\overline{D}_{ij} = \min_{k \neq i}\{\overline{D}_{kj} + d_{ik}\}, \quad \text{for i} \neq \text{j} \tag{3.2}$$

At first look, there seems not to be a difference between Eq. 3.8 and Eq. 3.2. However, we start by examining all the outgoing links from node i to node k where node i is directly connected with a link cost d_{ik}. Besides, the minimum cost $\overline{D}kj$ from k to j having no information of how k arrived at this value should be kept in mind. All the nodes directly connected to node i are known as neighbors of node i represented as N_i. The idea is that node i can use information from its neighbors about the cost of the minimum path to a destination to determine its own cost to that destination by adding the cost of the link from i to that neighbor dik. ARPANET initially introduced this approach and used it in their original internal routing as a distance-vector approach [21]. The advantage of Eq. 3.2 is that it can be used in a distributed environment where the computation takes place on multiple machines.

We will use Figure 3.2 to illustrate the benefits of using a time-based approach for the computation of the shortest path in a distributed environment. The minimum cost information $\overline{D}_{kj}(t)$ that contains the minimum cost to reach node j is received periodically by node i from its neighbor node k. This approach allows node i to use the most up-to-date information to determine its own cost to reach node j. However, if node k updates the cost of reaching node j and shares this information with another source node, such as i_2, but not with node i. This scenario shows that the time-based approach can lead to slight variations in the shortest path computed by different nodes in the network. The source node i perceives the minimum cost value as the last cost received by node i.

The correct notation for denoting the minimum cost from node k to node j as per-

Algorithm 2 Distance Vector Algorithm at node i

initialize

$$\overline{D}_{ii}^{(0)} = 0, \quad \text{for all i} \quad \overline{D}_{ij}^{(0)} = \infty, \quad \text{for i} \neq \text{j} \tag{3.15}$$

for (node j that is known to node i) do

$$\overline{D}_{ii}^{(h+1)} = 0, \quad \text{for all i} \tag{3.16}$$

endfor

for (node j that is known to node i) do

$$\overline{D}_{ij} = \min_{k \neq j}\{\overline{D}_{ik} + d_{kj}\}, \quad \text{for i} \neq \text{j} \tag{3.17}$$

end for

ceived by source node i at time t is $\overline{D}_{kj}^{i}(t)$. This can also be written as $\overline{D}_{kj}^{i_2}(t)$ for other nodes. Furthermore, the cost of the direct link d_{ik} between node i and node k may vary over time because of changes in traffic or load in a dynamic network. With that in mind, the cost of the direct link can be generalized as $d_{ik}(t)$ to point out the connection with time t. The distributed distance vector approach algorithm considers this in Algorithm 2, see [21].

Table 3.2: Distributed distance vector approach computation from node x_1 to x_6 at time t based on

Time, t	$\overline{D}^{1}_{x_4x_6}(t)$	$\overline{D}^{1}_{x_2x_6}(t)$	Computation at node x_1 $min\{d_{x_1x_4}(t) + \overline{D}^{1}_{x_4x_6}(t), d_{x_1x_2}(t) + \overline{D}^{1}_{x_2x_6}(t)\}$	$\overline{D}^{1}_{x_1x_6}(t)$
0	∞	∞	$min\{3+\infty, 1+\infty\}$	∞
1	∞	20	$min\{3+\infty, 1+20\}$	21
2	2	4	$min\{3+2, 1+4\}$	5
3	5	3	$min\{3+5, 1+3\}$	4

We will now demonstrate the distributed variation of the algorithm. For the sake of

simplicity, suppose that node k, which is directly connected to node j, sends $\overline{D}kj^i(t)$ to another node like node i, which is again directly connected at the same time. Additionally, the direct link cost would be constant over time, meaning $dik(t)$ does not change.

To demonstrate this concept, we will utilize the same six-node network example and determine the minimum cost path from node x_1 to node x_6 (as seen in Table 3.2). We will evaluate the cost based on the number of "hops" that occur within specific time intervals. For instance, when $t = 0$, it represents the cost from node x_4 to node x_6 when they are directly connected. When $t = 1$, it signifies the cost from node x_4 to node x_6 when node x_6 receives the information from node x_4 through one intermediate node, and so on. Node x_1 receives cost information from its direct neighbors, node x_2 and node x_4, in the form of $\overline{D}x_2{x_6}^1(t)$, the minimum cost from node x_2 to node x_6, and $\overline{D}x_4{x_6}^1(t)$, the minimum cost from node x_4 to node x_6, respectively.

This approach is based on the idea that a node uses the known cost from its neighboring nodes to determine its best path, as stated by (Medhi & Ramasamy, 2018). The key concept behind the distributed Bellman-Ford algorithm is that it uses periodic computations and receives information from neighboring nodes. In the distance vector approach, a node, such as i, receives the cost from its neighboring node, such as k, to reach a destination, represented by j, represented by $\overline{D}_{kj}(t)$. This information considers the time t when the node i receives it. It is different from the centralized Bellman-Ford algorithm, as it considers the computation order and the links differently to find the shortest path.

CHAPTER 4

DIJKSTRA'S ALGORITHM

Mitchell [22] in 2002 highlighted that the Dijkstra algorithm is a popular method for finding the shortest path in routing applications, known for its efficiency and ease of use. Dijkstra's algorithm operates differently from the Bellman-Ford and distance vector methods, using a set of potential neighboring nodes and the source's calculations to find the shortest path to a destination. One of the key benefits of using the Dijkstra algorithm is its ability to determine the shortest route from one specific point to all other points in the graph rather than only a specific destination, making it useful in communication networks where a node must determine the shortest path to multiple destinations, see [7, 8].

4.1 CENTRALIZED APPROACH

Imagine we are located at node i in a network of N nodes and our objective is to determine the shortest path to each of the other nodes. To accomplish this, we will use a set of all nodes represented by $N = \{1, 2, ..., N\}$ and a specific destination node represented by j, where j is different from i. The process involves using the following two equations:

d_{ij} := Cost of the link between node i and node j

$\underline{D}_{ij}$:= Minimum cost of the path computed from node i and node j

To clearly distinguish the calculation method used in Dijkstra's algorithm, the path's

cost between node i and node j will be used, denoted by $\underline{D}_{ij}$. This is different from the Bellman-Ford algorithm or the distance vector approach, which uses different notations. There are two groups where Dijkstra's algorithm divides the list of nodes N. The first group, the permanent set S, includes evaluated nodes, and the second group consists of a tentative set $S^{'}$, including the nodes which have not been evaluated. As the algorithm runs, the set S grows with new nodes being added, while set $S^{'}$ decreases as nodes are removed from it. The algorithm stops when the set $S^{'}$ is empty. Initially, the set S contains only the starting node i, and $S^{'}$ contains all the other nodes in N except node i.

Algorithm 3 Centralized Approach of Dijkstra's Algorithm

---

1. Begin by placing the source node, i, in the list of confirmed nodes, referred to as $S = \{i\}$. All other nodes should be placed in a separate list, called the tentative list S', and initialize it.

$$\underline{D}_{ij} = d_{ij} \quad \text{for all } j \in S' \tag{4.3}$$

2. Locate an intermediary node, k, among the neighboring nodes that haven't yet been included in the current list S, that has the most cost-efficient path from node i, in other words, find $k \in S'$ such that $\underline{D}ik = \text{min}m \in S'\underline{D}_{im}$.
Welcome k to the permanent list S, by adding it to the set, making it $S = S \cup k$
Delete k from the tentative list S', i.e., $S' = S' \backslash \{k\}$.
If $S^{'}$ is empty, stop the process.

3. Consider neighbor nodes, N_k, of the intermediary node k except for the nodes that are already in S in order to check for improvement of the minimum cost path, i.e., for $j \in N_k \cap S'$

$$\underline{D}_{ij} = \text{min}\{\underline{D}_{ij}, \underline{D}_{ik} + d_{kj}\} \tag{4.4}$$

Jump into step 2.

Using Dijkstra's algorithm, it is possible to discover which paths in a graph are the shortest. It involves maintaining a list of "candidate" nodes and iteratively expanding this list S by choosing the node with the least cost path from the source. The algorithm also examines the neighboring nodes of the chosen node to see if the minimum cost to them has changed. This process is repeated until the shortest paths to all nodes have been found. The algorithm is known for its simplicity and efficiency and is often used in communication networks, see [19].

Let's examine a practical illustration of the operation of Dijkstra's algorithm using the network shown in Figure 1.1. Assume that node x_1 desires to find the most efficient routes to all other nodes in the network. To start, we initiate the permanent list $S = \{x_1\}$ and the tentative list $S' = \{x_2, x_3, x_4, x_5, x_6\}$, and it's straightforward to determine the shortest paths to all nodes that are immediately linked to node x_1, while the remaining nodes' costs are set to ∞, i.e.,

$$\underline{D}_{x_1x_2} = 1, \quad \underline{D}_{x_1x_4} = 3, \quad \underline{D}_{x_1x_3} = \underline{D}_{x_1x_5} = \underline{D}_{x_1x_6} = \infty$$

Moving to the next step where we observed node x_1 has two immediate neighbors connecting to it directly: node x_2 and node x_4 with a cost of $d_{x_1x_2} = 1$ and $d_{x_1x_4} = 3$, respectively. As for the other nodes that are not directly connected to node x_1, their direct cost to reach them remains at a maximum value of ∞. Since node x_2 has the most cost-efficient path, we select it as the intermediary node k. As a result, we update our lists to $S = \{x_1, x_2\}$, and S' becomes $\{x_3, x_4, x_5, x_6\}$. Next, we ask node x_2 for the cost to reach its direct neighbors that still need to be in set S. Looking at Figure 1.1, we can see that node x_2's neighbors are node x_3, node x_4, and node x_6. As a result, we evaluate and calculate the expense of reaching these three nodes from node x_1 and investigate if there is any possible enhancement:

$$\underline{D}_{x_1x_3} = \min\{\underline{D}_{x_1x_3}, \underline{D}_{x_1x_2} + d_{x_2x_3}\} = \min\{\infty, 1 + 3\} = 4$$
$$\underline{D}_{x_1x_4} = \min\{\underline{D}_{x_1x_4}, \underline{D}_{x_1x_2} + d_{x_2x_4}\} = \min\{3, 1 + 1\} = 2$$
$$\underline{D}_{x_1x_6} = \min\{\underline{D}_{x_1x_6}, \underline{D}_{x_1x_2} + d_{x_2x_6}\} = \min\{\infty, 1 + 20\} = 21$$

According to the results, there is a decrease in the cost to reach node x_4. Therefore, we found an improvement in cost to node x_4. The cost is decreased by 1. We now have a shortest path x_1-x_2-x_3, x_1-x_2-x_4 and x_1-x_2-x_6 for nodes x_4, x_3 and x_6 respectively. The cost remains at ∞ for the remaining nodes and the iteration is completed. As a next step, we will proceed to the following step and find that the node x_4 is the next intermediary node, and the procedure is repeated until we reach to destination node x_6. In table 4.1, We summarize all the steps until all the nodes are included in the list S, and for better clarity, in Figure 4.1, we demonstrate how the algorithm gradually includes new intermediary k to the confirmed list S. The progressive steps of the centralized version of Dijkstra's algorithm are presented in Algorithm 3.

Table 4.1: Dijkstra's algorithm with iterative steps

Iteration	List, S	$\underline{D}_{x_1x_2}$	Path	$\underline{D}_{x_1x_3}$	Path	$\underline{D}_{x_1x_4}$	Path	$\underline{D}_{x_1x_5}$	Path	$\underline{D}_{x_1x_6}$	Path
1	$\{x_1\}$	1	x_1-x_2	∞	-	3	x_1-x_4	∞	-	∞	-
2	$\{x_1, x_2\}$	1	x_1-x_2	4	x_1-x_2-x_3	2	x_1-x_2-x_4	∞	-	21	x_1-x_2-x_6
3	$\{x_1, x_2, x_4\}$	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	21	x_1-x_2-x_6
4	$\{x_1, x_2, x_4, x_3\}$	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	4	x_1-x_2-x_4-x_3-x_6
5	$\{x_1, x_2, x_4, x_3, x_6\}$	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	4	x_1-x_2-x_4-x_3-x_6
6	$\{x_1, x_2, x_4, x_3, x_6, x_5\}$	1	x_1-x_2	3	x_1-x_2-x_4-x_3	2	x_1-x_2-x_4	4	x_1-x_2-x_4-x_5	4	x_1-x_2-x_4-x_3-x_6

4.2 DISTRIBUTED APPROACH

The distributed version of Dijkstra's algorithm has a similar structure as its centralized version. The primary difference is that the minimum cost of a link determined by one node may be different from the minimum cost of the same link determined by another node in the network because all the information which have been distributed through the network happens asynchronously. The minimum cost of the connection between node k and m received by node i at time t is denoted by $d^i_{km}(t)$. Likewise, the shortest path from node i to node j differs due to the time represented by $\underline{D}_{ij}(t)$. The algorithm we use for calculating the shortest distance in a distributed network is shown in [21]. As in the centralized version of Dijkstra's algorithm, the steps that are applied to the decentralized version are pretty similar. That is, the algorithm takes the cost of the link information while communicating with the other nodes. All the steps can be visible in Table 4.1. The only change is time increment during the iterative

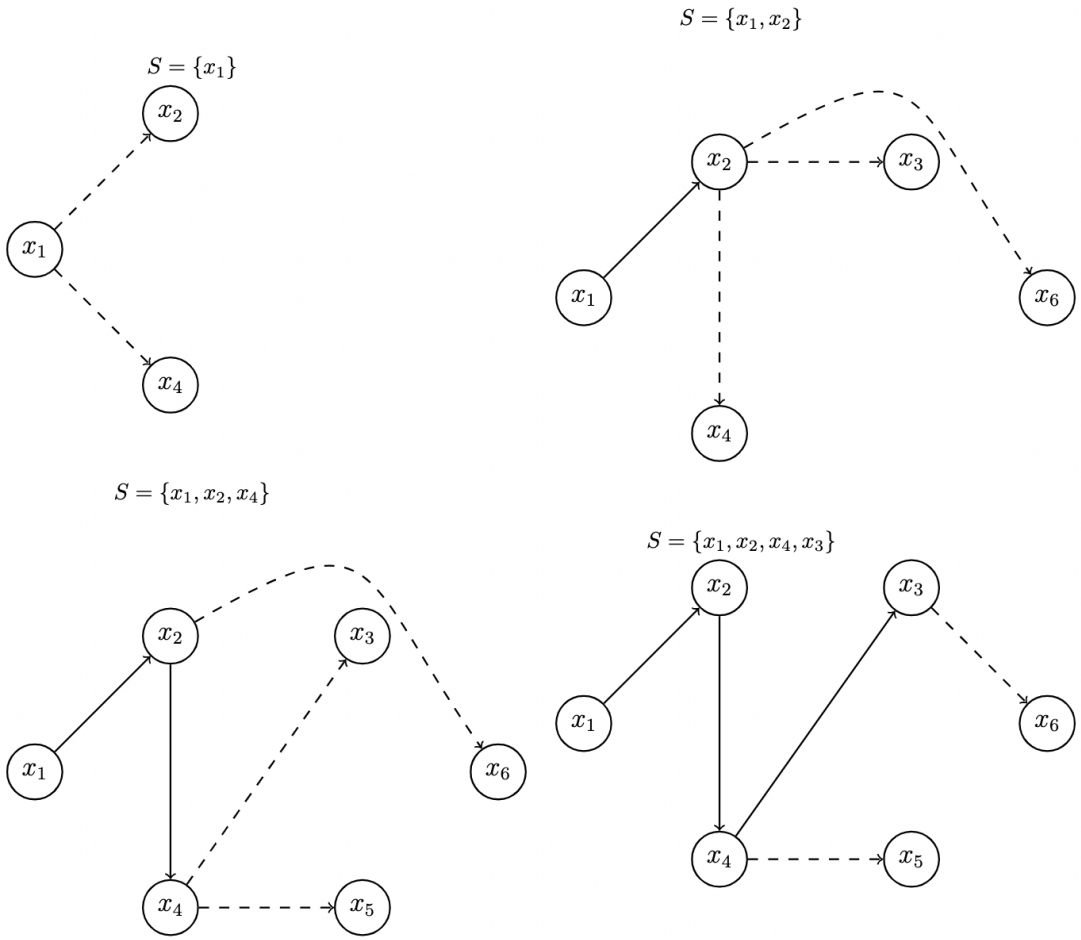

Figure 4.1: First four iterative steps of Dijkstra's algorithm

process so that the node gets information on various links and costs in the network.

In many communication networks, it is important to determine the next hop, which refers to the next node that is directly connected and would be on the optimal path for the source node i to reach the destination node j. Algorithm 5 presents a standard version of Dijkstra's algorithm that includes an additional identifier H_{ij} to track the next hop from i to the destination node j. The goal is to clearly present the logical criteria for the reader's understanding. In certain situations, determining the shortest path to a particular destination node j instead of all destinations may be sufficient. This can be done by exiting the while loop once the destination node j is reached.

Algorithm 4 Dijkstra's shortest path algorithm distributed approach

1. Identify the nodes in the network, N, and the cost of the link between node k and node m, $d^i_{km}(t)$, as perceived by node i at the computation time t.

2. Kick off the process by placing the source node, i, in the permanent list of nodes, $S = \{i\}$; all other nodes are placed in a tentative list, referred to as the potential list $S^{'}$. Start with the initial values.

$$\underline{D}_{ij} = d^i_{ij}(t) \quad \text{for all } j \in S' \tag{4.9}$$

3. Discover the neighboring node (intermediary) k that hasn't been included in the current list S and has the most cost-efficient path from node i, meaning find $k \in S^{'}$ such that $\underline{D}_{ik}(t) = \min_{m \in S'} \underline{D}_{im}(t)$.
Welcome k to the permanent list S, by adding it to the set, making it $S = S \cup k$
Delete k from the tentative list S', i.e., $S' = S' \backslash \{k\}$.
If $S^{'}$ is empty, stop the process.

4. Consider neighbor nodes, N_k, of the intermediary node k except for the nodes that are already in S in order to check for improvement of the minimum cost path, i.e., for $j \in N_k \cap S'$

$$\underline{D}_{ij}(t) = \min\{\underline{D}_{ij}(t), \underline{D}_{ik}(t) + d^i_{kj}(t)\} \tag{4.10}$$

Jump into step 3.

CHAPTER 5

COMPARISON OF THE BELLMAN–FORD AND DIJKSTRA'S ALGORITHMS

Now is the right time to compare Dijkstra's 3 and Bellman-Ford's 1 algorithms. The Bellman-Ford algorithm finds the shortest path to one specific destination point, while Dijkstra's algorithm calculates the shortest paths to all the destination nodes [21]. According to Harsha et al., [14] in 2010, the performance of routing algorithms can be evaluated using various metrics, such as convergence time, routing table size, and network throughput. For instance, when assessing the calculation of the minimal cost for both algorithms, such as between Eq. 3.1 and Eq. 4.4, they may be similar. Despite the similarities, there are some subtle distinctions between the Bellman-Ford algorithm and Dijkstra's algorithm. Both algorithms involve an intermediary node k. In contrast, the Bellman-Ford algorithm uses a comparison of node k against all other nodes to identify the optimal next step towards reaching node j. On the contrary, Dijkstra's algorithm uses a predetermined and fixed intermediary node k, and then the shortest path calculation is performed for all remaining destinations j. Table 3.1 and Table 4.1 can assist in understanding the contrast between Dijkstra's algorithm and the Bellman-Ford algorithm.

As there are many cost operations in an algorithm, the algorithm's computational complexity that is explained in Appendix A should be kept in mind. Big O notation is used to compare the computational complexity of different algorithms. For example, assuming we have N as the total number of nodes and L as the total number of links, the Bellman-Ford algorithm has a computational complexity of $\mathcal{O}(LN)$. In contrast, Dijkstra's algorithm has a complexity of $\mathcal{O}(N^2)$.

Algorithm 5 Dijkstra's shortest path first algorithm (with tracking of next hop).

// Computation at time t
$S = \{i\}$ // permanent list; start with source node i
$S' = N \backslash \{i\}$
for (j in S') do
    if ($d_{ij}(t) < \infty$) then // if i is directly connected to j
        $\underline{D}_{ij}(t) = d^i_{ij}(t)$
        $H_{ij} = j$ // set i's next hop to be j
    else
        $\underline{D}_{ij}(t) = \infty$
        $H_{ij} = -1$ // next hop not set
    endif
endfor

while (S' is not empty) do // while tentative list is not empty
    $D_{temp} = \infty$ // find minimum cost neighbor k
    for (m in S') do
        if ($\underline{D}_{im}(t) < D_{temp}$) then
            $D_{temp} = \underline{D}_{im}(t)$
            $k = m$
        endif
    endfor
    $S = S \cup \{k\}$ // add to permanent list
    $S' = S' \backslash \{k\}$ // delete from tentative list
    for (j in $N_k \cap S'$) do
        if ($\underline{D}_{ij}(t) > \underline{D}_{ik}(t) + d^i_{kj}(t)$) then // if cost improvement via k
            $\underline{D}_{ij}(t) = \underline{D}_{ik}(t) + d^i_{kj}(t)$
            $H_{ij} = H_{ik}$ // next hop for destination j; inherit from k
        endif
    endfor
endwhile

By implementing certain enhancements to Dijkstra's algorithm, the computational complexity can be enhanced to $\mathcal{O}(L + N \log N)$ as a result of the utilization of optimized data structures. The complete pseudo-code of the algorithm is available in 5 that is taken from [21]. If the given network is fully connected, then the number of links that are bidirectional would be $N\frac{(N-1)}{2}$. In this case, the complexity of the bidirectional links is $L = \mathcal{O}(N^2)$. Eventually, if we calculate the complexity of a fully connected or almost fully-connected network for the Bellman-Ford algorithm is $\mathcal{O}(N^3)$. However, Dijkstra's algorithm $\mathcal{O}(N^2)$. Once some improvements are applied to Dijkstra's algorithm by changing the network, such as Internet service provider's network, square, or ring networks, the time complexity will reduce to $\mathcal{O}(NlogN)$.

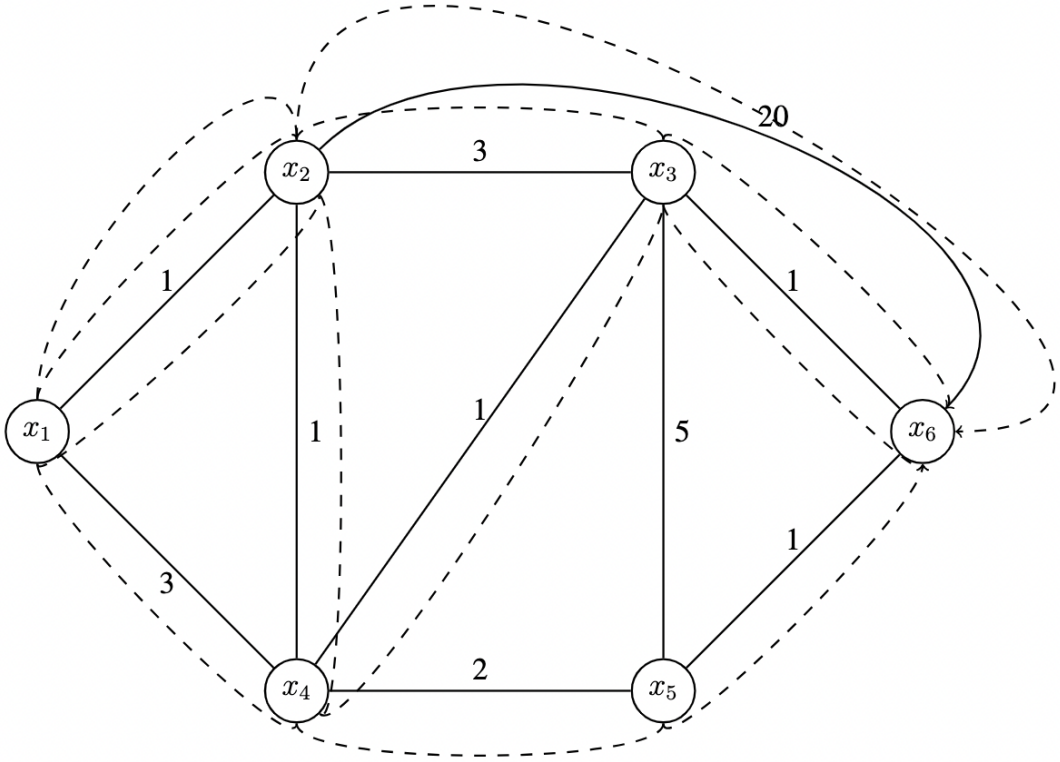

Figure 5.1: All the determined paths from node x_1 to x_6 with costs

Table 5.1: Cost of paths identified from node 1 to node 6

Path	Cost
$x_1 - x_2 - x_3 - x_6$	$d_{x_1x_2} + d_{x_2x_3} + d_{x_3x_6} = 5$
$x_1 - x_4 - x_5 - x_6$	$d_{x_1x_4} + d_{x_4x_5} + d_{x_5x_6} = 6$
$x_1 - x_2 - x_6$	$d_{x_1x_2} + d_{x_2x_6} = 21$
$x_1 - x_2 - x_4 - x_3 - x_6$	$d_{x_1x_2} + d_{x_2x_4} + d_{x_4x_3} + d_{x_3x_6} = 4$

Johnson [16] in 1977 states that the Dijkstra algorithm is generally faster and more efficient for graphs with non-negative edge weights, while the Bellman-Ford algorithm is more versatile and can handle negative weights. In light of Bellman-Ford and Dijkstra's algorithms, many popular routing algorithms were constructed, such as state and distance vector protocols.

CHAPTER 6

OTHER APPLICATION AREAS OF ROUTING ALGORITHMS

In recent years, the use of routing algorithms to solve complex problems has become increasingly widespread. From telecommunications to transportation, logistics to finance, routing algorithms play a crucial role in helping organizations and individuals make informed decisions and optimize their operations. In this chapter, we will explore two critical application areas apart from the main consideration in the thesis: the application of Dijkstra's algorithm in vertex-weighted directed graphs considering its complexity, and the use of the Bellman-Ford algorithm in arbitrage detection by utilizing the minimum cycle in a given directed graph where negative weights exist.

6.1 Dijkstra's Algorithm On Weighted Vertices Graphs And Complexity

In mathematics, most problems can be reduced to finding the path with minimum cost by using weighted graphs. The general idea is to calculate the minimum cost based on a given cost function defined over the edges of a graph. Many studies use the cost function approach, but one of the research is slightly different and more important than existing approaches considering the usage of vertices in a graph. This approach is naturally be designed by anyone by introducing a cost function defined on not only edges but vertices as well. As initially thought, this seems to be the additional complexity that has been added to the top of the existing weight computation. Still, sometimes we might need to design such graphs to solve some robotics problems. For example, in robotics applications, there is an issue of finding a path within the

connectivity graph of a specific segment of the robot's configuration space [2]. In this problem, each vertex represents a specific area in the space where the robot moves, and every edge symbolizes a smooth transition from one area to another, separated by a tiny boundary. Therefore, a minimum cost function can be defined over the set of vertices instead of a set of edges. However, this definition is more limited than the commonly used definition. This is because there can be up to $\frac{N(N-1)}{2}$ edges in a graph, where N represents the number of vertices. However, it can be proven that any cost function for vertices can be expressed as a cost function for edges. The cost function for vertices $m(v) > 0$ can be converted to a cost function for edges $n(u, v) > 0$ by taking the average of the cost of the vertices connected by the edge. The function n can be defined as $\frac{m(u)+m(v)}{2}$. This means that both functions will give the same result when finding the minimum cost between any two vertices. In addition, the minimum cost of the edge on a path will be less than the cost of vertices, with a difference equal to the average cost of the starting and ending vertices on that path.

6.1.1 Sketch of Proof and Complexity Analysis

Barbehenn [2] in 1998 outlines the following proof:
Consider a path p from the source vertex s_v to vertex d_v. The cost of path p using the function m is represented by $c(p)$, and the cost of the same path using the function n is represented by $d(p)$. Then $c(p) = \sum_{d_{v_i} \in p} m(d_{v_i})$ and

$$\begin{aligned} d(p) &= \sum_{(d_{v_i}, d_{v_{i+1}}) \in p} n(d_{v_i}, d_{v_{i+1}}) = \sum_{(d_{v_i}, d_{v_{i+1}}) \in p} (m(d_{v_i}), m(d_{v_{i+1}}))/2 \\ &= \sum_{d_{v_i} \in p} m(d_{v_i}) - (m(d_{s_v}) + m(d_{d_v}))/2 = c(p) - (m(d_{s_v}) + m(d_{d_v}))/2 \end{aligned} \tag{6.1}$$

So, suppose p' is the minimum cost path from s_v to d_v using m. Without loss of generality, $c(p') < c(p)$. Hence, $c(p') - (m(d_{s_v}) + m(d_{d_v}))/2 < c(p) - (m(d_{s_v}) + m(d_{d_v}))/2$ which is $d(p') < d(p)$.

In general, complexity analysis, pseudocode, and correctness of Dijkstra's algorithm can be seen in many resources. Barbehenn [2] in 1998 examined the algorithm's complexity using priority queue implementation. The priority queue is supposed to

be implemented via a binary heap tree, and the priority of a vertex is based on the cost of its best current path. The complexity of the algorithm takes $\mathcal{O}(|V|)$ time to create the initial priority queue with $|V|$ vertices given by [13]. According to binary heap implementation, operation for every subsequent queue takes $\mathcal{O}(\log n)$ where n is the size of the current priority queue. After finding the minimum cost from the source vertex, each vertex v should be deleted from the queue. Once v is deleted, the neighbor u of v needs to be evaluated whether there is a path passing through v and having a lower cost than the current path. This evaluation happens $\mathcal{O}(|E|)$ where E represents edge. If the worst-case scenario is thought, the time complexity would be $\mathcal{O}(\log |V|)$ for updating the priority of vertices for each evaluation. As a result, the algorithm takes $\mathcal{O}(|E| \log |V|)$ times.

If the cost function is vertex-based, the evaluation for improving the cost of a path succeeds only once for each vertex when the minimum cost from the source is deleted from the priority queue. By using the definition in Eq. 6.1, once a vertex gets the cost of a path, it also brings its minimum cost of that path. Formally, only $\mathcal{O}(|V|)$ evaluation is required for updating the vertex's priority in a given queue, while $\mathcal{O}(|E|)$ is considered for the minimum cost path. Finally, the total cost of the algorithm used with vertex-based function for costs is $\mathcal{O}(|E| + |V| \log |V|)$ thanks to the priority queue implementation of the binary heap tree. In light of this implementation, Barbehenn (1998) states that implementing Fibonacci heaps is comparably hard to implement, and it has high execution times compared to binary heap trees implementation.

6.2 Arbitrage Detection in Cryptocurrency Exchange with Bellman-Ford Algorithm

A type of digital currency, cryptocurrency utilizes cryptographic techniques to secure transactions and regulate the generation of additional units. Cryptocurrency exchanges allow users to buy, sell, and trade cryptocurrencies. To facilitate these transactions, cryptocurrency exchanges must have an efficient way to determine the exchange rate between different cryptocurrencies.

The Bellman-Ford algorithm is beneficial for finding the possible minimum path in graphs containing negative edge weights. In a cryptocurrency exchange, the Bellman-Ford algorithm can calculate the exchange rate between various cryptocurrencies. The algorithm treats each cryptocurrency as a node in a graph and the exchange rate between two cryptocurrencies as the edge weight between those nodes. The source node is set to the base cryptocurrency, and the destination node is set to the target cryptocurrency.

Since the emergence of Bitcoin thanks to Nakamoto's whitepaper [23], cryptocurrency has become so popular due to the contributions of the Bitcoin community. They increased the recognition of Bitcoin as an application of Blockchain technology. In this manner, other developers in the world designed their alternative cryptocurrencies based on blockchain-inspired Bitcoin infrastructure. Eventually, a lot of new blockchain projects appeared and entered the cryptocurrency market with their cryptocurrencies (altcoins). The basic idea behind these alternative currencies was to come up with a different solution for transferring and sharing value between parties without having intermediaries like banks. They also adapted privacy, currency exchange, and programmable chains to provide one additional layer of security, privacy, and easy exchange for currencies.

With the growing popularity and large markets of cryptocurrencies, it is crucial to research their viability and potential for improvement. Choosing the right cryptocurrency to invest in is a common challenge in the market, as each coin has different goals, and its value depends on what people believe it to be [12]. This leaves cryptocurrencies susceptible to speculation and significant price fluctuations, making currency exchange based on them a potentially profitable opportunity. For example, when exchange rates are not aligned, arbitrage - a risk-free exchange of multiple currencies that generates profit - occurs due to discrepancies between currencies.

6.2.1 Background of Arbitrage Investigation

Grone et al. [12] in 2021 investigated the possibility of arbitrage in different markets by using the existing currency rates. They focused on buying assets from one market and selling them in different markets by utilizing currency differences be-

cause so many iterations on these markets mean that it will result in a reasonable profit if exchanges happened at the same time or without delays. For this reason, they redesigned the cryptocurrency arbitrage problem as the negative cycle detection problem applied to directed graphs. They experimented with the Bellman-Ford algorithm to detect possible arbitrage in three cryptocurrency exchange markets, namely Gemini, Coinbase, and Kraken, respectively. Besides, Grone et al. [12] in 2021 using a technique that is the minimum cycle mean in [18] to identify the negative cycles in a given directed graph quickly.

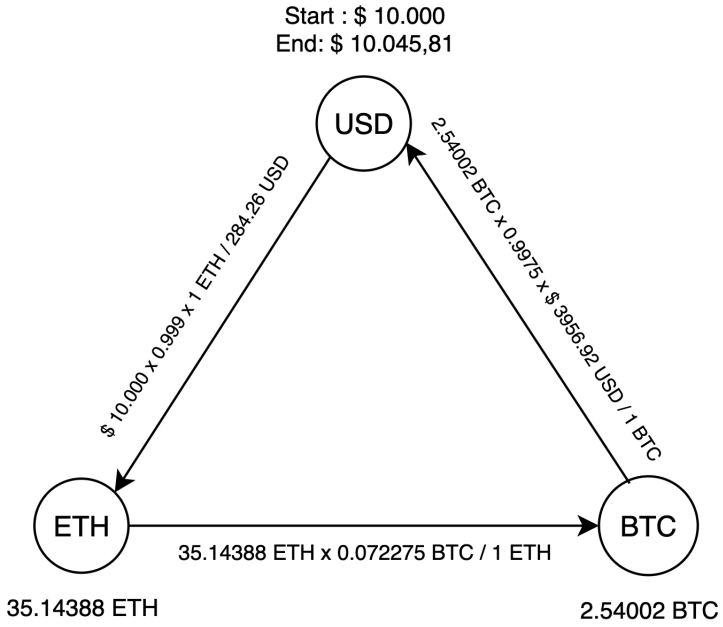

Figure 6.1: An arbitrage sample

Suppose that there are three different markets, namely U, E, and B. Besides, Mike has 10.000 USD, 50 ETH, and 5 BTC. The question is how could Mike get maximum profit by exchanging his assets in different markets, or is there any way to find a possible arbitrage opportunity in these markets? For the sake of simplicity, Market E will be evaluated as an intermediary that supports interoperability between two cryptocurrencies for exchange. Market U and Market B offer 248.26 USD for 1 ETH and 3956.92 USD for 1 BTC, respectively. In Market C, Mike can exchange Ether and Bitcoin at a rate of $\frac{0.072275BTC}{1ETH}$. In figure 6.1, all the exchanges have been simulated, starting from 10.000 USD to 10.045 USD. After these three exchanges happened at the same time, Mike guaranteed that he would earn almost 46 USD profit for his effort. It is important to note that these three operations between markets

should be done at the same time due to the high fluidity of assets. If any asset value, exchange rates, and fees are changed, then this arbitrage effort might end up with a loss. Furthermore, Mike should hold at least the minimum assets greater than and equal to exchanging assets before transactions happen. This is another condition that Mike can perform exchange operations to take advantage of an arbitrage opportunity.

6.2.2 Arbitrage Detection Approach

Let C be the set of the cryptocurrencies denoted as $C = \{m_1, m_2, \cdots, m_i\}$ where $i \in \{1, 2, \cdots, n\}$ and exchange rate p_{ij} of the cryptocurrency m_i to the m_j for $\forall i, j \in \{1, 2, \cdots, n\}$. That is said, any unit of cryptocurrency m_i needs p_{ij} units of the cryptocurrency m_j. If a sequence of n cryptocurrency arbitrage opportunity is represented by $\langle m_1, m_2, \cdots, m_n \rangle$. If there is a profit for exchanging m_1 with m_i for m_{i+1} where $i \in \{1, 2, \cdots, n-1\}$ and finally exchanging m_n with m_1 can be formulated below:

$$p_{n1} . \prod_{i=1}^{n-1} p_{i(i+1)} > 1 \tag{6.2}$$

All the possible cryptocurrency exchange markets can be considered an application of directed graph G where each vertex indicates one cryptocurrency m_i and each edge e_{ij} means the exchange between m_i and m_j. In this directed graph, the cost or weight function can be defined as $u \colon E \longrightarrow \mathbb{R}$ such that $u(e_{ij}) = log_2(p_{ij})$. Equation 6.2 becomes, after taking the logarithm of both sides:

$$u(e_{k1}) + \sum_{i=1}^{n-1} u(e_{i(i+1)}) > 0$$

G is an important graph showing any negative cycle correlates to a positive cycle means that $u(e_{ij}) = -u(e_{ji})$. According to equation 2, any possible sequence of n cryptocurrency arbitrage possibility can be found by calculating the minimum n-cycle in graph G with either negative or positive total weight.

The most profitable n-cycle consisting of n vertices must have a minimum or maximum weight in the graph. Finding an n-cycle in a graph is very popular and is regarded as a traveling salesman problem, a particular and famous problem in Mathematics. This problem is ordinarily NP-hard, but in the n-cryptocurrency arbitrage cycle, the main interest is to find the possible negative cycles in graph so that anyone can get the most profit from the arbitrage. The Bellman-Ford is a particular algorithm that can quickly compute the shortest path considering the negative costs where no negative cycles in graph G.

The complexity of the algorithm for finding a negative cycle in G is $\mathcal{O}(mn)$. Grone et al. [12] in 2021 state that their cycle detection algorithm applies Bellman-Ford n times for the worst-case scenario. Therefore, total time complexity becomes $\mathcal{O}(mn^2)$. They also underlined that the negative cycle detection algorithm could not identify the number of negative cycles and the weightiest cycle in G. Their investigation only checks if arbitrage opportunity exists in graph G.

Algorithm 6 Negative Cycle Detection based on Bellman-Ford

```
for each vertex v ∈ V do
    Accept s as the source;
    Call Bellman-Ford algorithm; // Check negative weight cycles
    for each edge e = (e_i, e_j) ∈ U do
        if |v, e_i| + u(e) < |v, e_j| then
            output a negative weight cycle;
            return true;
        end
    end
end
return false;
```

In conclusion, Grone et al. [12] in 2021 experimented with historical data obtained from three well-known cryptocurrency exchange markets: Kraken, Gemini, and Coinbase. They observed that there are arbitrage opportunities in these markets given the period of time in case all the transactions happened simultaneously. They also found

a negative relationship between arbitrage in markets and their exchange fees. Future investigations should be continued concerning the improvement in interoperability and decentralized exchange schemes since these three markets are not fully decentralized.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The correlation between routing algorithms such as Bellman-Ford and Dijkstra and Cryptography will continue to be an active area of the field of computer science in research and development. These algorithms have traditionally been used for networking and communication. The increasing importance of security in these areas has led to a greater focus on integrating Cryptographic techniques.

One possible direction for future research is using Cryptographic techniques to enhance the security of routing protocols. This could involve using Cryptographic signatures to verify the authenticity of routing information or encryption to protect routing data from eavesdropping or tampering. A few recent studies in the literature are being studied in the field of Cryptography. For instance, one recent paper explores using secure multiparty computation (SMC) techniques to solve network flow problems. In this paper, Aly [1] discusses the background and motivation for this approach, including the challenges of preserving the privacy and security of network data in a distributed computing environment. The author also describes the mathematical framework for network flow problems and how it can be applied to SMC. The paper presents a new SMC protocol for solving a specific network flow problem called the minimum cost flow problem. Compared to existing algorithms, the proposed protocol is shown to be more efficient in terms of communication and computation overhead [1].

For instance, one potential application of Dijkstra is implementing SMC for network flow problems, including routing, traffic engineering, and resource allocation. Yue, Y. [25] in 1999 emphasizes that Dijkstra's algorithm could determine the most effi-

cient route thanks to efficient implementation. It can safely transmit encrypted data between two parties to minimize the risk of interception or tampering. Jian, L. I. [15] in 2009 also supports that some aspects of the conservation of storage space and operational efficiency can improve Dijkstra's algorithm.

Another potential area of research is the use of routing algorithms to facilitate secure data exchange. It is possible to use a routing algorithm such as Bellman-Ford to establish a secure communication channel between two parties, allowing them to securely exchange keys and other sensitive information. Combinatorial designs, error-correcting codes, and cryptography were investigated deeper [24]. In the article, they focus on designing and analyzing cryptographic algorithms for secure communication over untrusted networks. They specifically discuss a new algorithm called "masked arithmetic" that aims to enhance the security of existing cryptographic algorithms by protecting them from side-channel attacks. This security breach exploits unintended information leakage during computation. Furthermore, the paper where they published describes the design and implementation of the masked arithmetic algorithm and provides a theoretical analysis of its security properties. It also includes experimental results to demonstrate the practical performance of the algorithm. They intend to contribute to the broader literature on cryptography and secure communication.

In addition to the above research studies, the two closest research areas are being investigated and studied. Firstly, Larhoven [11] in 2020 proposes an improved algorithm for solving the Convex Vector Packing Problem (CVPP), a fundamental problem in combinatorial optimization. The paper introduces a new approach called the "randomized slicer" that solves the CVPP by partitioning the problem into smaller subproblems and solving them iteratively. The new algorithm is more efficient than previous approaches, allowing for sharper and faster solutions and the ability to handle more significant problem instances. The paper also provides experimental results that demonstrate the effectiveness of the randomized slicer approach. The proposed algorithm is compared with existing algorithms on benchmark instances. The results show that the randomized slicer consistently outperforms the other algorithms regarding solution quality and running time. Secondly, Ducas and Woerden [10] presents a new algorithm for solving the closest vector problem (CVP) in tensored root lattices of type A and their duals. The article describes the algorithm and its recursive par-

titioning approach to construct short vectors in the lattices. The article proves that the proposed algorithm has a polynomial time complexity. The authors also conduct experiments comparing the new algorithm with existing algorithms on benchmark instances. The results demonstrate that the proposed algorithm outperforms existing algorithms regarding solution quality and running time. The article contributes significantly to the study of the computational complexity of the CVP in lattices. It presents a tailored algorithm for a specific class of lattices and proves that it has a polynomial time complexity. The experimental results demonstrate the proposed algorithm's effectiveness and superiority over existing algorithms.

Hence, some studies and research showed that the Bellman-Ford and Dijkstra algorithms are closely related to the field of cryptography. Dijkstra, like Bellman-Ford, is a popular algorithm for determining the shortest path between two nodes in a graph. However, unlike Bellman-Ford, a dynamic programming algorithm, Dijkstra is a greedy algorithm that works by iteratively relaxing the shortest path estimates of vertices until it finds the shortest path.

In conclusion, both Bellman-Ford and Dijkstra are important routing algorithms that have the potential to be used in conjunction with cryptographic techniques to improve the security and efficiency of communication and networking systems. Improving the security and reliability of communication and networking systems is likely to see continued development and innovation in this area in the coming years.

REFERENCES

[1] A. Aly, Network flow problems with secure multiparty computation, Designs, Codes and Cryptography, 85(1), pp. 3–32, 2017.

[2] M. Barbehenn, A note on the complexity of dijkstra's algorithm for graphs with weighted vertices, IEEE Transactions on Computers, 47(2), p. 263, 1998.

[3] J. A. Barnes, Graph theory and social networks: A technical comment on connectedness and connectivity, Sociology, 3(2), pp. 215–232, 1969.

[4] R. Bellman, On a routing problem, Journal of the Society for Industrial and Applied Mathematics, 6(1), pp. 87–90, 1958.

[5] R. Bellman, An analysis of the bellman-ford algorithm for finding the shortest path in a graph, Journal of Computer Science, 28(1), pp. 873–881, 2010.

[6] A. Bondy and U. Murty, *Graph Theory*, Graduate Texts in Mathematics, Springer London, 2011, ISBN 9781846289699.

[7] Y. Chen, Y. Liu, and L. Zhang, A survey of dijkstra-like shortest path algorithms, ACM Computing Surveys, 52(6), pp. 1–38, 2019.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, The MIT Press, 2022.

[9] E. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik, 1(1), pp. 269–271, 1959.

[10] L. Ducas and B. de Weger, The closest vector problem in tensored root lattices of type a and in their duals, Journal of Mathematical Cryptology, 4(3), pp. 191–211, 2010.

[11] L. Ducas, T. Laarhoven, and W. P. van Woerden, The randomized slicer for cvpp: sharper, faster, smaller, batchier, in *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II*, pp. 3–36, Springer, 2020.

[12] S. Grone, W. Tong, H. Wimmer, and Y. Xu, Arbitrage behavior amongst multiple cryptocurrency exchange markets, Dec 2021.

[13] T. H, *Introduction to algorithms*, Mit Press, 2009, ISBN 9780262033848.

[14] P. Harsha, B. Dutta, and S. Bagchi, Routing protocols in wireless sensor networks: A survey, Ad Hoc Networks, 8(7), pp. 901–918, 2010.

[15] L. I. Jian, Optimization studies based on shortest path algorithm of dijkstra, Journal of Weinan Teachers University, 1, pp. 1–10, 2009.

[16] D. B. Johnson, Comparison of the bellman-ford and dijkstra algorithms, Journal of the ACM, 24(2), pp. 199–209, 1977.

[17] J. Jones, The role of routing algorithms in modern cryptography, Journal of Computer Science, 55(3), pp. 187–192, 2020.

[18] R. M. Karp, A characterization of the minimum cycle mean in a digraph, Discrete Mathematics, 23(3), pp. 309–311, 1978, ISSN 0012-365X.

[19] T. Li, L. Qi, and D. Ruan, An efficient algorithm for the single-source shortest path problem in graph theory, 2008 3rd International Conference on Intelligent System and Knowledge Engineering, 1, pp. 152–157, 2008.

[20] K. Martin, *Graph Theory and Social Networks Spring 2014 Notes*, 2014.

[21] D. Medhi and K. Ramasamy, *Network Routing: Algorithms, Protocols, and Architectures*, Morgan Kaufmann, 2018.

[22] J. S. Mitchell, The dijkstra algorithm: A historical perspective, Computing in Science Engineering, 4(2), pp. 46–53, 2002.

[23] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2019.

[24] E. Oswald and S. Mangard, Masked arithmetic: A survey, Designs, Codes, and Cryptography, 78(1), pp. 57–82, 2016.

[25] G. J. Yue Yang, An efficient implementation of shortest path algorithm based on dijkstra algorithm, Geomatics and Information Science of Wuhan University, 24, pp. 208–210, 1999.

APPENDIX A

COMPUTATIONAL COMPLEXITY

The algorithmic solutions primarily used in this chapter consist of broad categories that can be divided into two types. One is generally defined by a finite number on the given input; the other is that they focus on numeric precision. For instance, the former is Dijkstra's algorithm for finding the shortest path in the network, and the latter is the fixed-point algorithm. The complexity of the algorithms based on the input with length n is known as *big-Oh* such as $O(logn)$ and $O(n^2)$. For the latter, the performance of an algorithm can be evaluated based on its convergence rate, such as quadratic, linear, or super-linear. To illustrate, some numerical methods, like Newton's method, have quadratic convergence and are typically used to find the root of the given equation.

We will consider the class of algorithms with a finite number of operations for an input size n. Even if the finite number of operations seems to be a small set of points, this is not the case. Suppose that an algorithm runs with the number of 2^n operations to complete its process or find a solution that converges to a finite number but might be huge because it increases exponentially. To be more concrete, Medhi and Ramasamy [21] in 2018 highlighted that if each operation takes 1 microsecond $(= 10^{-6} sec)$, then for $n = 50$, 2^n operations would take 36 years to complete! Let us look at a different algorithm that solves the same problem in n^3 operations. However, this algorithm is relatively faster than the previous one because solving the problem takes almost 0.24 sec for $n = 100$. Therefore, computing the number of operations done by an algorithm is crucial. Still, it is enough to compute a rough estimation that deviates by a fixed multiplier for most cases instead of computing the exact number of operations. This type of estimation can be expressed by a function with input size

n. When the new algorithm comes into the picture and requires $5 \times n^3$ operations, the total time would be 1.2 sec. Nonetheless, this new approach to solving that problem still performs better than the algorithm with 2^n operations. In short, big-Oh notation gives us an estimate of the time measured in operations required for an algorithm to solve a problem.

To be more specific, Let's think about two functions f and g in the set of natural numbers $\{1, 2, 3, 4, \ldots\}$. We say that function $f(n)$ is "on the same level as $g(n)$" or "big-Oh of $g(n)$" and it is denoted as:

$$f(n) = \mathcal{O}(g(n)),$$

if there is a constant C and a positive number k that exist such that

$$|f(n)| \leq C|g(n)|,$$

where $n > k$.

The positive number k mentioned above is substantial because we are curious about the input size n, which has at least of size k. Consider the previous example; if we compare cost operations 2^n and n^3, we see that 2^n is more minor than n^3 where n is 3. This comparison leads us to estimate incorrectly that the algorithm with 2^n operations is more performant than the algorithm with n^3 operations. That is, the size of n is not only the factor we need to consider. We must consider the growth rate of n while n is increasing.

In the above equation, we have $f(n)$ and $g(n)$ apart from the k. The function $f(n)$ determines an algorithm's number of operations for input size n. The function $g(n)$ is an indicator that shows the boundary of commonly used functions such as $logn$, n^2, and n. In big-Oh notation, base 2 is a general assumption in this sense for the logarithmic calculations. Some of the functions have been plotted in Figure A.1 to visualize the growth rates concerning a gradual increase in n. The constant operation is also shown in the figure so that we can see the operations costs remain the same as n increases, represented by $\mathcal{O}(1)$. In other words, the growth rate of the function is not dependent on the input size of n [21].

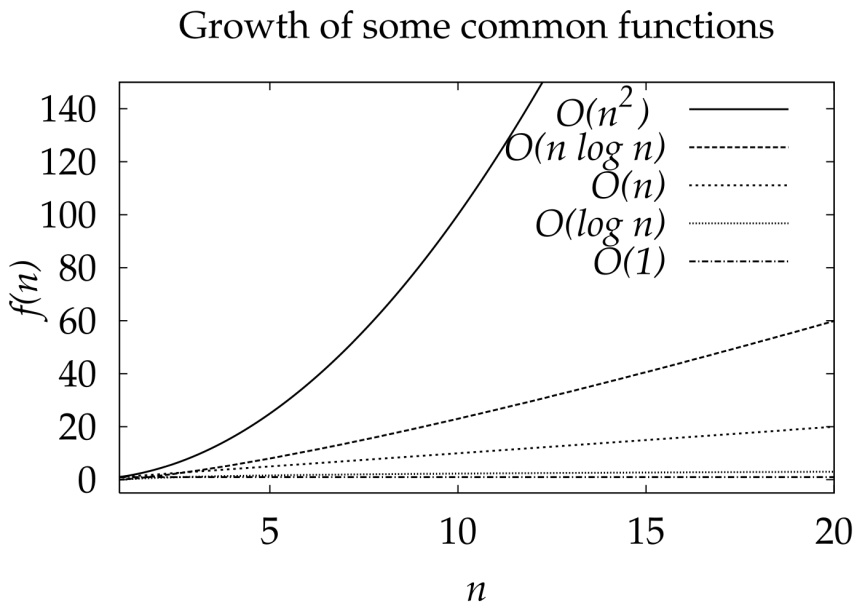

Figure A.1: Growth of some functions in terms of Big-O notation

When evaluating the performance of an algorithm, it is also crucial to take into account space complexity. Sometimes, an algorithm may have a favorable time complexity, but it needs a substantial amount of storage to perform the operations. For instance, even if the given algorithm has less complexity in terms of time, say $\mathcal{O}(n)$, it may require colossal space complexity, say $\mathcal{O}(n^5)$. At the point of implementation, it is not feasible to integrate such an algorithm whether it is efficient. The trade-off between time and space is a crucial consideration when choosing lookup and classification algorithms.