An Optimization Problem of Internet Routing

by

Jiaxin Liu

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2014

© Jiaxin Liu 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Abstract

As the Internet usage grows, existing network infrastructure must deal with increasing demand. One way to deal with this is to increase network capacity, and another, is to set network parameters appropriately. In this dissertation we contribute to the latter approach by determining the unique network paths data must flow over from its origin to its destination, while accounting for an Active Queue Management method, Random Early Detection (RED). We formulate a mixed integer non-linear program to determine the data paths, referred to as a routing policy. We prove that determining an optimal routing policy that accounts for RED is NP-Hard. Furthermore, in order for the generated routing policies to be real-world implementable, also known as realizable, we must determine weights for all arcs in the network such that solving the all pairs shortest path problem using these weights reproduces the routing policies. We show that determining if our generated routing policies are realizable is NP-Hard. Fortunately, using traffic data from three real-world networks, we are able to find realizable routing policies for these networks that account for RED, using an off-the-shelf solver, and policies found perform better than those used in those networks at the time the data was collected.

Acknowledgements

I would like to thank my supervisors, Professor Laura Sanità, Professor Stan Dimitrov for all the advices throughout my two years of master's study. They contributed plenty of key ideas presented in my thesis, and shared their experiences with me to prevent me from unnecessary effort. Laura was always able to answer my questions and concerns quickly, while being patient all the time. Stan taught me to always pay attention to detail and pushed me in improving my efficiency of work. Without their help, I hardly believe that I can overcome the obstacles of transitioning from an undergraduate to a master student and successfully complete this thesis.

I would like to thank Professor James H. Bookbinder and Professor Ricardo Fukasawa for being my readers and giving me feedback.

I would like to thank Prof. Yin Zhang with putting his collection of Abilene Network traffic data publicly available. Thank CANARIE's staff Mr. Mark Wolff, Mr. Thomas Tam, and Mr. Jun Jian for sharing traffic data with us. Thank Steven Gunn for the latex template us this thesis from .

Finally, I would like to thank all of my friends to vitalize my spare time and share advices about graduate studies. I would like to thank my family and my dear P for their constant support.

Contents

List of Figures

Chapter 1

Introduction

Today's world is witnessing the fast growth of the Internet while embracing the benefits it brings. Over the past decade, Internet usage has increased by nearly 40% year by year [7], and according to an estimate by Cisco [8], this upward trend is likely to carry on over the next few years at a consistent pace. As much as people enjoy the convenience offered by the vast development of the Internet, they have to occasionally contend with slow speeds, especially when networks fail to accommodate large demands, resulting in network congestion [23, 33]. To resolve the slow speeds resulting from congestion, network service providers might use network provisioning, usually associated with hardware upgrades. However, network provisioning may require investment in network infrastructure, and this process could be very costly. Alternatively, network operators may reconfigure the protocols used by the existing infrastructure to obtain a better traffic distribution, thereby reducing the degree of congestion without further network investment. In this dissertation, we show one method that can indeed deliver more traffic in a network simply by reconfiguring existing network protocols. Specifically, we discuss a mathematical model that maximizes the amount of data traffic delivered in a network under a congestion control mechanism while accounting for the behaviour of standard network protocols.

We model a computer network as a simple graph that contains nodes to represent routers and arcs to represent links. Based on this graphical model, we develop a mathematical optimization program, inspired by the classical multi-commodity flow problem [21], to model the network traffic flow process. This model finds routing policies, i.e., the paths that the data must take from its source to destination, that maximize the total data received at all destinations, while accounting for Random

Early Detection (RED), to deal with congestion. RED is a protocol implemented on each router that increasingly drops incoming data as the flow of data across that router increases. In conjunction with other network protocols, RED is able to control the data transmission rate and hence reduces network congestion. Our model determines the routing policies that take into account the data dropping mechanism of RED, and indeed, our model minimizes data loss from such dropping mechanism by maximizing the total data delivered. We show that the developed model is computationally intractable. However, using an off-the-shelf solver we can obtain good solutions to real-world network instances [1]. The results of our experiments show that our generated policies perform better than the policies used by the real-world networks. We also develop a revised formulation that provides robust routing policies that account for different demand realizations, and the results of the corresponding experiments show that the generated robust routing policies outperform the policies used in real-world networks. In addition, we examine the realizability problem for generated routing policies; that is, we would like to determine whether routing policies generated by our mathematical programs can be configured using standard routing protocols, so that these policies are real-world applicable. In particular, we consider the Open Shortest Path First (OSPF) routing protocol [32], where OSPF requires paths be the same as those found by solving an all-pairs shortest path problem, or simply put, the paths must be the shortest with respect to a set of pre-determined arc weights. We show that we can find OSPF arc weights for the corresponding network graph for each of our instances in the experiments even though, in general, this problem is computationally intractable, one thing that is also shown in this document.

To summarize the contributions of this work is as follows:

- Define the mathematical model to find the all-pairs routing policies that account for RED.
- Prove that the model above is NP-Hard.
- Formulate the all-pairs inverse shortest path problem, used to determine OSPF arc weights that realize a prescribed set of paths as shortest paths.
- Prove that all-pairs inverse shortest path problem is NP-Hard.
- Show using the models above, we can generate realizable all-pairs OSPF routing policies for 3 real-world networks, and the ones found are better than those currently used.

This dissertation is organized as follows. Chapter 2 introduces the background and context of our research, as well as the related work. Chapter 3 formulates the mathematical model for network traffic flow under RED and shows the properties associated with the model. Chapter 4 discusses practical implication of the model from Chapter 3. Chapter 5 gives a proof of NP-Hardness of the All-Pair Routing Problem Under RED defined in Chapter 3. Chapter 6 gives a proof of the NP-Hardness of the All-Pair Inverse Shortest Path problem defined in Chapter 3. Finally, Chapter 7 summarizes the project and discusses future directions.

Chapter 2

Background

In this chapter, we introduce the background of our work, focusing on the structure of computer networks and the network protocol stack. We then go through the recent development in two fields that relate to our work, intra-domain traffic engineering and Random Early Detection. Finally, we include the related work in these two fields.

2.1 Network Layers and Protocols

A classic computer network is designed with a multi-layer structure, using protocols to dictate data transmission among layers [31]. Networking model is defined differently in the literature so that there exist various layer stacks. Our discussion below is based on the definition in Kurose and Forouzan [31]. The physcial layer and the link layer deals with the physical components and link to link hop. The application layer provides network applications to network users who generate data requests. In addition, lying in between the link layer and the application layer, the transport layer and the network layer are designed to regulate data transmission behaviour with various types of protocols, thereby making connections between the link and the application layers. Protocols in the transport layer, e.g., TCP and UDP, define the end-user data transmission, i.e., how data should be sent between end user processes; protocols in the network layer define the end-to-end data transmission process across a network between end user machines. When all the protocols of these two layers interacting with each other, network users send and request data, thereby generating data traffic flow across the network.

There are many network protocols in both the transport layer and the network layer for various purposes. Network administrators may even slightly modify standard protocols for specific needs. As a result, networks from different regions may use different sets of protocols that work better for their own interests. A set of routers administered by one organization is referred to as an autonomous system (AS). Within an AS, routing policies are set in accordance with the interests of the organization [22, 30, 31, 34]. In an AS, network administrators use intra-domain routing protocols to determine paths between each source and destination router pair; a forwarding table is built in each router, indicating the next router that data is sent. In particular, shortest path routing protocols, for example Open Shortest Path First (OSPF), are one of the commonly used routing protocols. These protocols determine the shortest paths to transmit data between the source and destination routers, with respect to certain arc metrics. If we consider an AS as a simple graph, where each router is a node and each communication link is an arc, then on each arc, network administrators assign an upper-bounded positive integer (in OSPF [32] case bounded above by $2^{16}-1$) as the administrative weight. Moreover, the routing protocol used by this AS chooses a shortest path algorithm (e.g. Dijkstra's Algorithm [19]) to determine the shortest path with respect to these administrative weights.

2.2 Intra-Domain Traffic Engineering

In general, configurations of routing protocols tend to be stable. For example, according to Cisco's suggestion [6], OSPF arc weights can be set as the reciprocals of arc capacities. Arc capacities are unlikely to change, unless hardware upgrades are made; therefore, Cisco's standard arc weights are likely to remain constant. This stability may seem reliable in normal situations, but it does not guarantee good network performance, which is important especially when network usage increases. Typical network performance measures include average delay, packet loss percentage, and total throughput. As more traffic is generated due to the boost in network usage, it could be preferable for network operators to reconfigure routing protocols so that some of the performance measures are optimized. This idea helps motivate the invention of Traffic Engineering. Traffic Engineering (TE) [11, 12] is the process that provides decisions in the construction and administration of a network by using the measurement, modelling, application and control of Internet traffic in order to optimize network

performance measures. In particular, the intra-domain TE problem deals with intra-domain routing protocols as noted earlier in this chapter. Many researchers studied the intra-domain TE problem and have focused specifically on variants where shortest path routing protocols are used [10, 14, 26]. In order to restrict the model under the context of the shortest path routing protocols, the inverse shortest path problem [17] are incorporated as part of the problem. The inverse shortest path problem determines a set of arc weights such that some prescribed paths are shortest paths with respect to these weights. Even though most of these TE problems are proven to be hard to solve, they are still able to perform well for certain real-world situations. Some national research networks have used relevant results from solving their TE problems in designing their routing policies for a few years [15].

Intra-domain TE problem under shortest path routing has been studied extensively in the past decade [14, 26]. The survey done in 2009 [10] by a group of outstanding researchers in this field has a thorough breakdown of this field. Fortz and Thorpe gives a formulation based on the multicommodity flow problem [21] and prove the NP-Hardness of the TE problem with Equal Cost Multipath rule [26], and discuss a local-search heuristics. Bley focus on the unique shortest path rule [14], and prove the NP-Hardness of the inverse shortest path problem for unique shortest path rule as well as the routing problem. We prove a hardness result similar to Bley's inverse shortest path problem and used a similar structure to Bley's proof.

2.3 Random Early Detection

One of the major concerns by network engineers, as well as the entire Internet community, is congestion, which occurs when there is too much data traffic sent through the same transmission link. When a network is congested, depending upon the underlying transport layer protocols, different scenarios may occur. Slow speeds are common outcomes; while in a more severe scenario, data may have to be dropped out of the network to revive a heavily congested link. Congestion control has been an increasingly popular topic of network design. In the transport layer, Transmission Control Protocol (TCP) has its own congestion control procedure. Whenever a data packet is dropped, a TCP sender infers a packet is dropped and slows down the sending rate exponentially and resends the packet. However, as pointed out in [16], this congestion control mechanism reacts too slowly to a burst of traffic flow coming into a congested node. Based on the standard Tail Drop rule [18] for queue management, the

burst of flow will fill up a limited-sized buffer, where all incoming packets are waiting to be processed at the router, and all the remaining unqueued data packets will be dropped. Even though under TCP, every dropped data packets is guaranteed to be resent, Tail Drop mechanism could still result in irrecoverable outcomes. In order to provide a more effective congestion control method, Active Queue Management (AQM) was developed and standardized from early 2000s [16]. One of the earliest AQM, Random Early Detection (RED), is widely considered and extended over the past two decades [25]. RED specifies two parameters, *maxth*, the maximum threshold, and *minth*, the minimum threshold, corresponding to the maximum and the minimum buffer size for RED mechanism respectively. When the queue length is less than the minimum threshold, RED retains all incoming data packets in the queue; when the queue length is larger than the maximum threshold, all the incoming packets will be dropped. Whenever the queue length is within *maxth* and *minth*, each newly arrived packet is dropped with a known probability that increases with queue length. The RED process is summrized in Figure 2.1. The difference between RED and Tail Drop is that RED can possibly drop packets even when the buffer is not full while Tail Drop will only drop packets from the queue whenever it is full. Clearly, RED prevents bad results when there are bursts of traffic flow approaching routers, a scenario that usually occurs in practice [16].

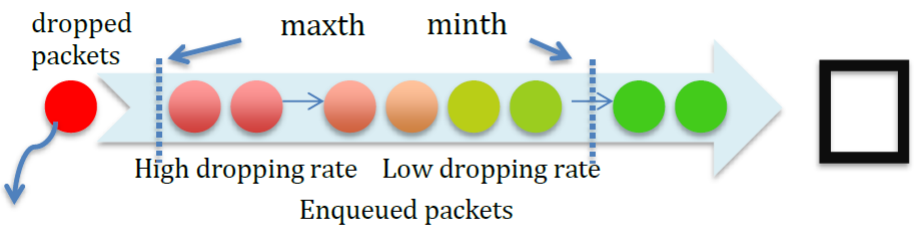

FIGURE 2.1: Random Early Detection

Random Early Detection has been studied since mid-1990s when it was introduced by [25]. The Internet Engineering Task Force (IETF) has provided documents formalizing and standardizing RED [16] Many researchers analyze the shortcomings of the original RED and invent their respective variants. Most of the researches regarding the variants of RED have taken the queueing analysis into account, with simulation results shown, in order to understand the analytical behavior of RED and the possible enhancement of RED [9, 13, 24]. Cisco uses a variant of RED called Weighted RED

(WRED), which is a slight modification of RED with a weight component to adjust the thresholds for different traffic flow classes [3, 4]. Dimitrov [20] introduces a deterministic way of describing RED mechanism, and incorporate it into a TE problem with Multi-protocol Label Switching (MPLS). The deterministic RED function invented by Dimitrov measures the proportional data flow received over data flow sent, and use the corresponding data loss across an arc to simulate the dropping process of RED. We incorporate RED mechanism into the TE formulation and search for the optimal OSPF routing policy.

Chapter 3

Modelling

In this Chapter, we formally introduce the main routing problem to address, the OSPF routing problem under RED. We reexamine RED from a mathematical perspective, and develop functions that approximate the RED process. To solve the routing problem, we propose and explain a mathematical program that incorporates the RED functions. We discuss the inverse shortest path problem used to determine arc metrics based on the derived routing policy from the math program. Finally, we show several properties associated with the models.

3.1 RED functions

We consider Random Early Detection (RED) introduced in Chapter 2. Recall that to implement RED, we need to specify two parameters, *minth* and *maxth*, of buffer length. According to RED, all incoming packets will be dropped when the queue length exceed *maxth*. Note that in a traditional Drop-Tail mechanism, incoming packets are dropped when the buffer is full, i.e., the queue length reaches the capacity of the buffer. Due to this similarity, we can think of the *maxth* as the effective capacity, or buffer size, of the buffer and therefore render the *maxth* irrelevant. Let u be *maxth*, the effective capacity, and β be the *minth*, the effective starting point of data dropping. Then $\beta \leq u$. Let t_{in} be the variable representing the current queue length in the buffer, then the surviving probability of the data packet is 1 if $t_{\text{in}} \leq \beta$ and 0 if $t_{\text{in}} \geq u$. Note that if we set $\beta = u =$ capacity then this probability function is a binary function that depicts the Drop-Tail mechanism.

When $\beta \leq t_{\text{in}} \leq u$, any incoming data packet is assigned with a probability, whose value is determined by the queue length. RED assumes that the relationship between the dropping probability and the queue length is linear. With the notations and assumptions above, we define a function $g : \mathbb{Z}^+ \to [0, 1]$ of queue length, to represent the surviving probability as follows:

$$g(t_{\text{in}}) = \begin{cases} 1 & 0 \leq t_{\text{in}} \leq \beta \\ 1 - \frac{t_{\text{in}} - \beta}{u - \beta} & \beta \leq t_{\text{in}} \leq u \\ 0 & u \leq t_{\text{in}} \end{cases},$$

Note that the domain of the surviving probability function is all positive integers, since we are counting the number of packets in the queue. It is much more convenient to extend the domain into all the real numbers, so we have to tweak some concepts here. We define traffic flow as the size of the data packets transmitted. If the flow over an arc is 5GB, it means that the size of all the packets that pass through this arc is 5GB. Therefore, we can use the traffic flow as a continuous variable describing the amount of traffic coming into a buffer. Thus, we have a extended function $g : \mathbb{R}^+[0, 1]$ of traffic flow, to represent the proportion of data traffic retained due to RED.

Note that the function g determines the retained data according to the flow received t_{in} (enqueued packets). However, it is more convenient to express the retained data in terms of the flow sent over the arc, t_{out}. We introduce a function to quantify the percentage of data retained over an arc in terms of t_{out}. Consider an arc (i, j). Specifically, suppose we send t_{out} units of traffic flow over (i, j) from i. Note that the buffer at j is not accepting all t_{out} units of flow, due to dropping. We define a function $f : \mathbb{R} \to [0, 1]$ of flow sent from the head node, to represent the proportion of flow received at the tail node. With this function, we see that $t_{\text{out}} \cdot f(t_{\text{out}})$ represent the amount of flow that is enqueued at node j, i.e., t_{in}. By definition, $g(t_{\text{in}}) = g(t_{\text{out}} \cdot f(t_{\text{out}}))$ is the proportion of flow received at node j, which equals to $f(t_{\text{out}})$. Thus we have

$$\begin{aligned} f(t_{\text{out}}) &= g(t_{\text{out}} f(t_{\text{out}})) \\ &= \begin{cases} 1 & 0 \leq t_{\text{out}} f(t_{\text{out}}) \leq \beta \\ 1 - \frac{t_{\text{out}} f(t_{\text{out}}) - \beta}{u - \beta} & \beta \leq t_{\text{out}} f(t_{\text{out}}) \leq u \\ 0 & u \leq t_{\text{out}} f(t_{\text{out}}) \end{cases}. \end{aligned} \tag{3.1}$$

Note that since $f(t_{\text{out}}) \leq 1$, $t_{\text{out}} f(t_{\text{out}}) \leq \beta$ if $t_{\text{out}} \leq \beta$, so $f(t_{\text{out}}) = 1$ for all $0 \leq t_{\text{out}} \leq \beta$. Also when $t_{\text{out}} \geq \beta$, it satisfies that $f(t_{\text{out}}) = 1 - \frac{t_{\text{out}} f(t_{\text{out}}) - \beta}{u - \beta}$, and solving the equation for $f(t_{\text{out}})$ gives $f(t_{\text{out}}) = \frac{u}{u - \beta + t_{\text{out}}}$. Moreover, as t_{out} grows, $f(t_{\text{out}})$ decreases, and $t_{\text{out}} f(t_{\text{out}})$ will never be greater than u, therefore, $f(t_{\text{out}})$ will never be zero. Thus, we have the following close-form formula for f:

$$f(t_{\text{out}}) = \begin{cases} 1 & \text{if } 0 \leq t_{\text{out}} \leq \beta \\ \frac{u}{u - \beta + t_{\text{out}}} & \text{if } \beta \leq t_{\text{out}} \end{cases}.$$

3.2 Problem Definition

We define the All-Pairs routing problem under RED as follows.

Given: Let $G = (N, A)$ be a directed graph, T be the set of all-pair commodities such that $|T| = N \cdot (N - 1)$. For each Commodity $k \in T$, let c^k be the weight, $s^k > 0$ be the units of flow at the source node $o^k \in N$ to be sent to the destination node $d^k \in N$.

Find: A unique simple path $\{a^k\} \subseteq A$ for each $k \in T$.

Objective: Maximize the total weighted flow delivered under RED.

Considering an Internet backbone network, we assume that there is positive demand across all pairs of nodes in the network. Therefore in the problem definition, there are $N \cdot (N - 1)$ commodities to indicate that every ordered pair induces a commodity, and $s^k > 0, \forall k \in T$ indicates that all the commodities have positive demands.

We show that this problem is NP-Hard, deducing that the original OSPF Routing Problem under RED is NP-Hard.

Theorem 3.1. *All-Pair OSPF Routing Problem under RED is NP-Hard.*

The proof is shown in Chapter 5.

3.3 The Mathematical Program

We define the following variables and notations:

- x_i^k: the amount of flow of commodity $k \in T$ present at node $i \in N$
- α_{ij}^k: binary variables indicating whether commodity $k \in T$ is send through arc (i, j), for each arc $(i, j) \in A$
- f_{ij}: the RED function f as in (3.1), for arc $(i, j) \in A$

We propose a model to solve the All-Pairs routing problem under RED as follows:

$$\max_{\alpha,x} \quad \sum_{k=1}^{K} c^k x_{d^k}^k \tag{3.2a}$$

$$\text{s.t.} \quad x_{o^k}^k = s^k \qquad k \in T \tag{3.2b}$$

$$\sum_{j \in \delta^+(o^k)} \alpha_{o^k j}^k = 1 \qquad k \in T \tag{3.2c}$$

$$\sum_{j \in \delta^-(d^k)} \alpha_{j d^k}^k = 1 \qquad k \in T \tag{3.2d}$$

$$\sum_{j \in \delta^-(o^k)} \alpha_{j o^k}^k = 0 \qquad k \in T \tag{3.2e}$$

$$\sum_{j \in \delta^+(d^k)} \alpha_{d^k j}^k = 0 \qquad k \in T \tag{3.2f}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k = \sum_{i \in \delta^+(j)} \alpha_{ji}^k \qquad k \in T, j \neq o^k, d^k \tag{3.2g}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k \leq 1 \qquad k \in T, j \neq o^k, d^k \tag{3.2h}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k x_i^k f_{ij}(\sum_{l=1}^{K} \alpha_{ij}^l x_i^l) - x_j^k = 0 \qquad k \in T, j \in N : j \neq o^k \tag{3.2i}$$

$$\alpha_{ij}^k \in \{0, 1\}^n \qquad (i, j) \in A, k \in T \tag{3.2j}$$

.

The Source Constraints (3.2b) represent the value of the flow units at origin node for each commodity. The four constraints (3.2c), (3.2d), (3.2e), (3.2f) ensures that there will be exactly one arc out of the origin selected and one arc into destination selected

for a fixed commodity, and a commodity does not return to its source or leaves from its destination. The Balance Constraints (3.2g) guarantees the outflow and inflow of each commodity is the same for all non source and destination nodes . The next constraint, (3.2h), ensures there is at most one arc used to send each commodity out of each non source and destination node in the network, and it guarantees that there will not be any cycles (notice that α may induce some node disjoint cycles, but these cycles are disconnected with either the origin or destination of any commodity and therefore are ignored).

In summary, the constraints ensure there is one simple path between each origin-destination pair as the routing path and data flow is sent through these paths subject to RED, and the total weighted flow is maximized.

3.4 Non-convexity of (3.2)

We say a mathematical program convex if its objective function is convex and its feasible region is convex, that is, the feasible region contains the mid points between any two distinct feasible points. We show that (3.2) is non-convex.

Theorem 3.2. *The feasible region of the "relaxed" Model*

$$(3.2b)(3.2c)(3.2d)(3.2e)(3.2f)(3.2g)(3.2h)(3.2i) \tag{3.3a}$$

$$\alpha_{ij}^k \geq 0 \qquad (i,j) \in A, k \in T \tag{3.3b}$$

is non-convex.

Proof. Consider (3.3) on Diamond Network shown in Figure 3.1 with two units of one commodity to send from Node 1 to Node 4, and the RED function is

$$f_{i,j}(t) = \begin{cases} 1 & \text{if } 0 \leq t \leq 1 \\ \frac{2}{1+t} & \text{o.w.} \end{cases}.$$

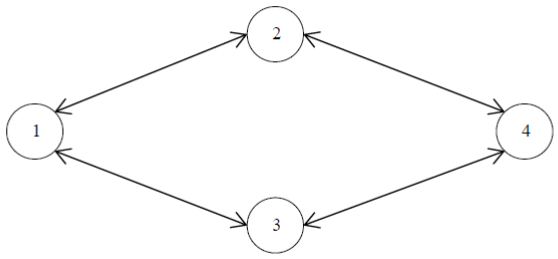

FIGURE 3.1: Diamond Network

There are two possible paths that form two feasible solutions:

$$\begin{aligned} P_1 &: \text{Node 1} \rightarrow \text{Node 2} \rightarrow \text{Node 4} \\ P_2 &: \text{Node 1} \rightarrow \text{Node 3} \rightarrow \text{Node 4} \end{aligned}.$$

The corresponding solutions are determined as follows: with 2 units of flow in the first step, $\frac{2}{2-1+2} = \frac{2}{3}$ of the flow will be preserved. At the second step, the flow received will be $2 \times \frac{2}{3} = \frac{4}{3}$, so $\frac{2}{2-1+\frac{4}{3}} = \frac{6}{7}$ will be preserved, thus, the total flow received at the destination node is $\frac{4}{3} \times \frac{6}{7} = \frac{8}{7}$.

$$\begin{aligned} &(x_1^1, x_2^1, x_3^1, x_4^1, \alpha_{12}^1, \alpha_{24}^1, \alpha_{13}^1, \alpha_{34}) \\ P_1 : \gamma_1 &\equiv (2, \frac{4}{3}, 0, \frac{8}{7}, 1, 1, 0, 0) \\ P_2 : \gamma_2 &\equiv (2, 0, \frac{4}{3}, \frac{8}{7}, 0, 0, 1, 1) \end{aligned}.$$

However, the average $\overline{\gamma} \equiv \frac{1}{2}(\gamma_1 + \gamma_2)$ is not a feasible solution to (3.3). The α components of $\overline{\gamma}$ is $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ but the corresponding flow component x should be $(2, 1, 1, 2)$, due to the fact that we separate the flow evenly with 1 through two paths and therefore no loss is incurred in the entire network, and clearly which does not equal to the x components of $\overline{\gamma}$.

Therefore, the feasible region of (3.3) is non convex. □

We prove that the feasible region of (3.3) is non-convex and hence conclude that (3.2) is a non-convex mixed integer non-linear programming problem. To further extend our analysis, we will show that this problem is in fact NP-Hard, to conclude that there are unlikely efficient algorithms to solve the problem exactly. On the other hand, at

the current stage, we aim to obtain a feasible solution, i.e., a routing scheme that might be more promising than the industry benchmark.

3.5 OSPF Path Realizability

Since our ultimate goal is to find an improved OSPF Routing policy, we need to verify the computed routing policy from (3.2) is indeed OSPF-configurable, or OSPF-realizable. In general, this is a decision problem in which given a set of simple paths, we decide if there is a set of arc metrics with which the paths can be realizable. This problem is called the Inverse Shortest Path (ISP) problem. ISP has been studied from early 1990s [17]. Bley [14] shows how to apply ISP to network application.

The formal statement of Inverse Shortest Path (ISP) is as follows:

Given: A graph $G = (N, A)$, and a set of paths $P_k, k \in T$.

Find: arc weight $\lambda_a \in \mathbb{N} \mid \lambda \geq 1, \forall a \in A$ s.t. the given paths are the shortest paths w.r.t. λ.

The Shortest Path Problem, the opposite of ISP, is one of the earliest optimization problems in graph theory. Algorithms based on dynamic programming or linear optimization are studied [19]. The problem is defined as follows:

Definition 3.3 (r-s Shortest Path Problem)**.** Given a digraph $G = (N, A)$ and arc weight $\lambda \in \mathbb{R}_+^{|A|}$, and two distinct node $s, r \in N$. Determine the shortest rs-path with respect to the weight λ.

Note that $\forall a \in A, \exists$ ordered pair $(u, v) = a$, so we also use notation (u, v) to denote an arc. Let $x_e, \forall e \in A$ be the indicator of whether e is in the shortest r-s path. Then the shortest r-s path linear program is as follows:

$$\begin{aligned} \min_x \quad & \sum_{e \in A} \lambda_e x_e \\ \text{s.t.} \quad & \sum_{(w,v) \in A} x_{(w,v)} - \sum_{(v,w) \in A} x_{(v,w)} = b_v, \forall v \in N, \\ & x_{(v,w)} \geq 0, \forall (v, w) \in A \end{aligned} \tag{P-SP}$$

where

$$b_v = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = r \\ 0 & \text{otherwise} \end{cases}.$$

The dual can be found as follows:

$$\begin{aligned} \max_{y} \quad & y_s - y_r \\ \text{s.t.} \quad & y_w - y_v \leq \lambda_{(v,w)}, \forall vw \in A \end{aligned}. \tag{D-SP}$$

According to the Complementary Slackness Condition, we know that if x, y are optimal for (P-SP) and (D-SP) respectively,

$$x_{vw} > 0 \Rightarrow y_w - y_v = \lambda_{(v,w)}.$$

Therefore, suppose x is fixed, a necessary condition for x being the characteristic vector of a shortest path is that we can find $y \in \mathbb{R}^N$, $\lambda \in \mathbb{N}_+^A$ s.t.

$$y_w - y_v = \lambda_{(v,w)}.$$

Thus, we can define for each node $v \in N$ a vector $y^u \in \mathbb{R}^N$, and formulate the Inverse Shortest Path problem as follows:

$$y_w^u - y_v^u \leq \lambda_{(v,w)}, \forall u \in N, \forall (v,w) \in A \tag{3.4a}$$

$$y_w^u - y_v^u = \lambda_{(v,w)}, \forall u \in N, u = o^k, (v,w) \in P_k, \forall k \in T \tag{3.4b}$$

$$\lambda \geq 1, \lambda \in \mathbb{N}^A \tag{3.4c}$$

, where P_k in (3.4b) is the prescribed path for commodity k.

Note that if we allow λ to take values from all non-negative real numbers, we can just set $\lambda = 0$ to be the trivial solution and the problem is solved. Therefore, we want to set the weight to be positive and for practical purpose, we let $\lambda \geq 1$.

Note that (3.5) is polynomially solvable when all the parameters are rational, because we can solve it as a linear program and round the solution to integer values. However,

in the context of network routing, as arc weights used by shortest path routing protocols, such as OSPF and IS-IS, are bounded above, so we need to add an additional constraint to accommodate this feature, obtaining the following formulation:

$$(3.4a)(3.4b) \tag{3.5a}$$

$$\lambda \geq 1, \lambda \in \mathbb{N}^A, \lambda \leq D \tag{3.5b}$$

, where D is the upper bound of the arc weights

Most of the inverse shortest path formulation in the literature is based on the one above, where $y_v^u, u \in N, v \in N$ is interpreted as the weight of the shortest path from u to v. Many researchers [14, 27] in the field of network design have studied this problem to help understand better the OSPF path realizability problem. Bley proved an inapproximability result for inverse shortest path problem with unique shortest path. We looked at a revised version of this problem such that a path between any pair of nodes are prescribed as the shortest path while the paths do not have to be unique.

As in the context of our research, we consider backbone networks where routing policies for all pairs of nodes need to be found, we also need to consider the All-Pairs variant of the Inverse Shortest Path Problem, which to our knowledge has yet to be addressed in the literature.

3.5.1 All-Pair Inverse Shortest Path Problem

Similar to the All-Pair OSPF Routing under RED, we also study the All-Pairs variant of Inverse Shortest Path Problem where the inputs are paths between every pair of nodes. We show that the variant of this problem where the paths are not necessarily unique shortest paths is NP-Hard. Work by Bley [14] looks at the ISP variants for directed network and commodities are a subset of all pairs of nodes.

Given: a graph $G = (V, E)$, where V is the vertex set and E is the edge set, and a set of paths of G: $\{P_{ij} \subset E : i, j \in V, i \neq j\}$, a positive number D.

Find: edge weights $\lambda_e \in \mathbb{N}_+, e \in E$, s.t. $\lambda_e \leq D$, and P_{ij} is the shortest path between i and j. (The shortest paths are not necessarily unique.).

Theorem 3.4. *The All-Pair Non-Unique Inverse Shortest Path Problem is NP-Hard.*

The proof is shown in Chapter 6. Our proof is inspired by Bley's proof. We adapt a similar graph structure that Bley invented.

3.6 Possibility of Merging the Routing Problem and the Realizability Problem

We study the All-Pairs Routing Problem under RED and the All-Pairs Inverse Shortest Path Problem separately and examine properties associated with the individual problems. We use the paths generated by (3.2) as input in the (3.5) to see if the paths are realizable. It should be noted that it is not necessary to separate the All-Pairs Routing Problem under RED and the All-Pairs Inverse Shortest Path Problem. We could incorporate (3.5) constraints into (3.2) and solve the combined problem. However, combining the formulation does necessarily introduces additional integer variables in the order of $|A|$ and therefore can potentially increase the runtime. Furthermore, we find that it might not be necessary to restrict α in (3.2) with the ISP constraints, as our experiments show that all the computed routing policies from the All-Pairs Routing Problem are realizable. The details of the experiments are shown in Chapter 4.

Chapter 4

Numerical Experiments

Even though the All-Pairs Routing Problem under RED is NP-hard, in this chapter, we show how we can solve the problem for 3 real-world networks. Before presenting our results, we show the modifications made to (3.2) in order to improve its solvability.

4.1 Modification of the formulation

We illustrate a few approaches in modifying (3.2) in this section.

4.1.1 Approximating the RED function

Recall that the RED function defined in (3.1) is a continuous, piecewise function. Since $f(t) \leq 1$ for all $t > \beta$, we can rewrite the function as a capped function, i.e.,

$$f(t) = \min\left\{1, \frac{u}{u - \beta + t}\right\}, t \geq 0.$$

In general, the original RED function (3.1), being a non-smooth function, brings computational difficulty to the problem. Preliminary results suggest that the problem scales exponentially, and for a medium sized network with 11 nodes and 14 arcs, most solvers cannot even stop iterating in weeks. To solve this computational issue, we propose to approximate the RED function with a smooth functions, with which the modified model can still produce a feasible routing policy (α as in (3.2)).

The approximated RED function, denoted by $f^\star(t)$, should satisfy the following three properties:

1. $f^\star$ is a smooth function with close form formula,
2. $f^\star$ should be less than or equal to 1 to preserve its probabilistic definition,
3. $f^\star$ should be a strictly decreasing function and converges to 0 when $t \to \infty$.

We define

$$f^\star(t) = \frac{u-\beta}{u-\beta+t}, \tag{4.1}$$

which satisfies all three properties above.

Thus, we can formulate the approximated model, Model II as follows:

$$\max_{\alpha,x} \quad \sum_{k=1}^{K} c^k x^k_{d^k} \tag{4.2a}$$

$$\text{s.t.} \quad \sum_{i\in\delta^-(j)} \alpha^k_{ij} x^k_i f^\star_{ij}(\sum_{l=1}^{K} \alpha^l_{ij} x^l_i) - x^k_j = 0 \quad k \in T, j \in N : j \neq o^k \tag{4.2b}$$

$$(3.2b)(3.2c)(3.2d)(3.2e)(3.2f)(3.2g)(3.2h)(3.2j) \tag{4.2c}$$

.

With the approximated RED function defined above, the runtime of most solvers is reduced significantly, from weeks to hour, and thus we are able to obtain feasible solutions for many larger instances. We use (4.2) to only determine α. Treating the determined α as a parameter, we use (3.2) to determine the corresponding x and use that x to evaluate the performance of (4.2). As α is already determined by (4.2) we can rewrite (3.2) as:

$$\max_{x} \quad \sum_{k=1}^{K} c^k x^k_{d^k} \tag{4.3a}$$

$$\text{s.t.} \quad x^k_{o^k} = s^k \qquad k \in T \tag{4.3b}$$

$$\sum_{i\in\delta^-(j)} \alpha^k_{ij} x^k_i f_{ij}(\sum_{l=1}^{K} \alpha^l_{ij} x^l_i) - x^k_j = 0 \qquad k \in T, j \in N : j \neq o^k \tag{4.3c}$$

4.1.2 Tree Constraints

In addition to addressing the computational issues of the RED functions of (3.2), we must help with the realizability, finding corresponding arc weights, given α. We add tree constraints into (4.2) in order to determine arc weights. To accommodate the new tree constraints, we define the new variables

$$\beta_{ij}^u = \begin{cases} 1 & \text{if some path rooted from } u \text{ uses arc } (i,j) \\ 0 & \text{o.w.} \end{cases}, u \in N, (i,j) \in A.$$

We add the tree constraints to (4.2) to have the formulation used in our experiments:

$$\max_{\alpha,\beta,x} \quad \sum_{k=1}^{K} c^k x_{d^k}^k \tag{4.4a}$$

$$\text{s.t.} \quad x_{o^k}^k = s^k \qquad k \in T \tag{4.4b}$$

$$\sum_{j \in \delta^+(o^k)} \alpha_{o^k j}^k = 1 \qquad k \in T \tag{4.4c}$$

$$\sum_{j \in \delta^-(d^k)} \alpha_{j d^k}^k = 1 \qquad k \in T \tag{4.4d}$$

$$\sum_{j \in \delta^-(o^k)} \alpha_{j o^k}^k = 0 \qquad k \in T \tag{4.4e}$$

$$\sum_{j \in \delta^+(d^k)} \alpha_{d^k j}^k = 0 \qquad k \in T \tag{4.4f}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k = \sum_{i \in \delta^+(j)} \alpha_{ji}^k \qquad k \in T, j \neq o^k, d^k \tag{4.4g}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k \leq 1 \qquad k \in T, j \neq o^k, d^k \tag{4.4h}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k x_i^k f_{ij}(\sum_{l=1}^{K} \alpha_{ij}^l x_i^l) - x_j^k = 0 \qquad k \in T, j \in N : j \neq o^k \tag{4.4i}$$

$$\alpha_{ij}^k \leq \beta_{ij}^u \qquad u \in N, o^k = u \tag{4.4j}$$

$$\sum_{(i,j) \in A} \beta_{ij}^u = |N| - 1 \qquad u \in N \tag{4.4k}$$

$$\alpha_{ij}^k \in \{0,1\} \qquad (i,j) \in A, k \in T \tag{4.4l}$$

$$\beta_{ij}^u \in \{0,1\} \qquad (i,j) \in A, u \in N \tag{4.4m}$$

.

The constraints (4.4j) describe the relationship between the α and β variables, and together with the tree constraints (4.4k) they ensure that every subgraph induced by shortest paths rooted from each node is a spanning tree.

4.2 Experiments and Results

For the experiments, we use Bonmin [1] as the solver, as it is one of the few solvers available that are able to handle non-linear, non-convex math programs and obtain a feasible solution in relatively short amount of time. Like many other off-the-shelf solvers, Bonmin ensures optimality for convex optimization but does not guarantee optimality for non-convex problems; however, as we will see, it produces a relatively "good" solution for these problems.

4.2.1 Networks

Our experiments use data from 3 real-world networks, all of which are National Research and Education Networks (NREN). NREN is the backbone network dedicated to research and education institutes; the routers in the network are typically located at major cities and universities. Abilene Network is the Unites States' NREN prior to 2007, when it was retired and upgraded into the current "Internet2 Network". Taiwan Advance Research and Education Network (TWAREN) [5] is the current Taiwanese NREN. Canada's Advanced Research and Innovation Network (CANARIE) [2] is the current Canadian NREN.

Abilene Network contains 11 nodes and 14 edges; TWAREN contains 13 nodes and 20 edges; CANARIE contains 7 nodes and 11 edges. The networks are illustrated in Figure 4.1, Figure 4.2 and Figure 4.3.

For each network, we take the hourly traffic demands for a one-week period as input and solve (4.4) for each. For Abilene, traffic demands between all pairs of nodes are collected, so we directly use them. For TWAREN and CANARIE, we estimate the traffic demands using link utilization. The α in the solutions are then used in (4.3) to determine the objective value. We then use (3.5) to check whether the obtained paths induced by α are realizable. The objective values over the 2 weeks of instances are

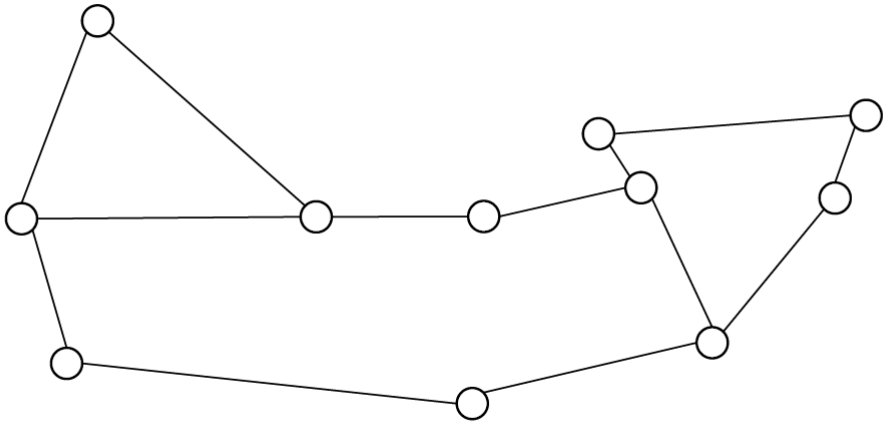

Figure 4.1: Abilene Network

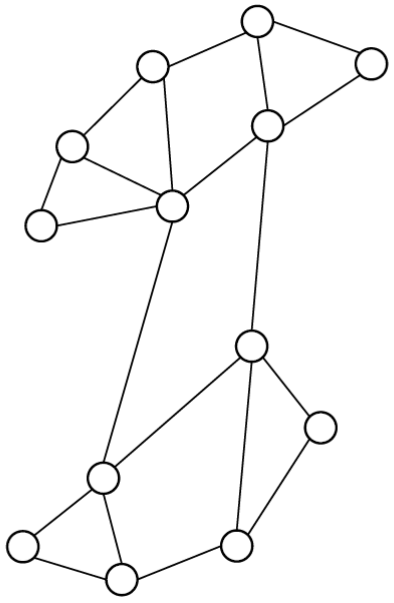

Figure 4.2: TWAREN

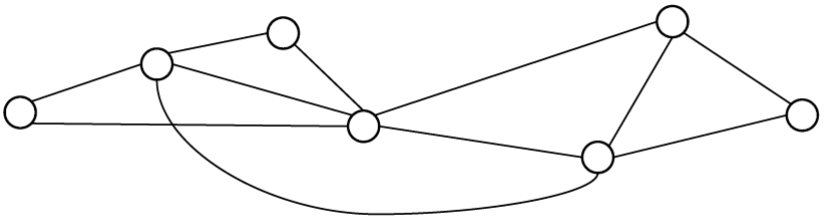

Figure 4.3: CANARIE

compared with those using the industry standard routing policies and the comparison is illustrated in the following graph.

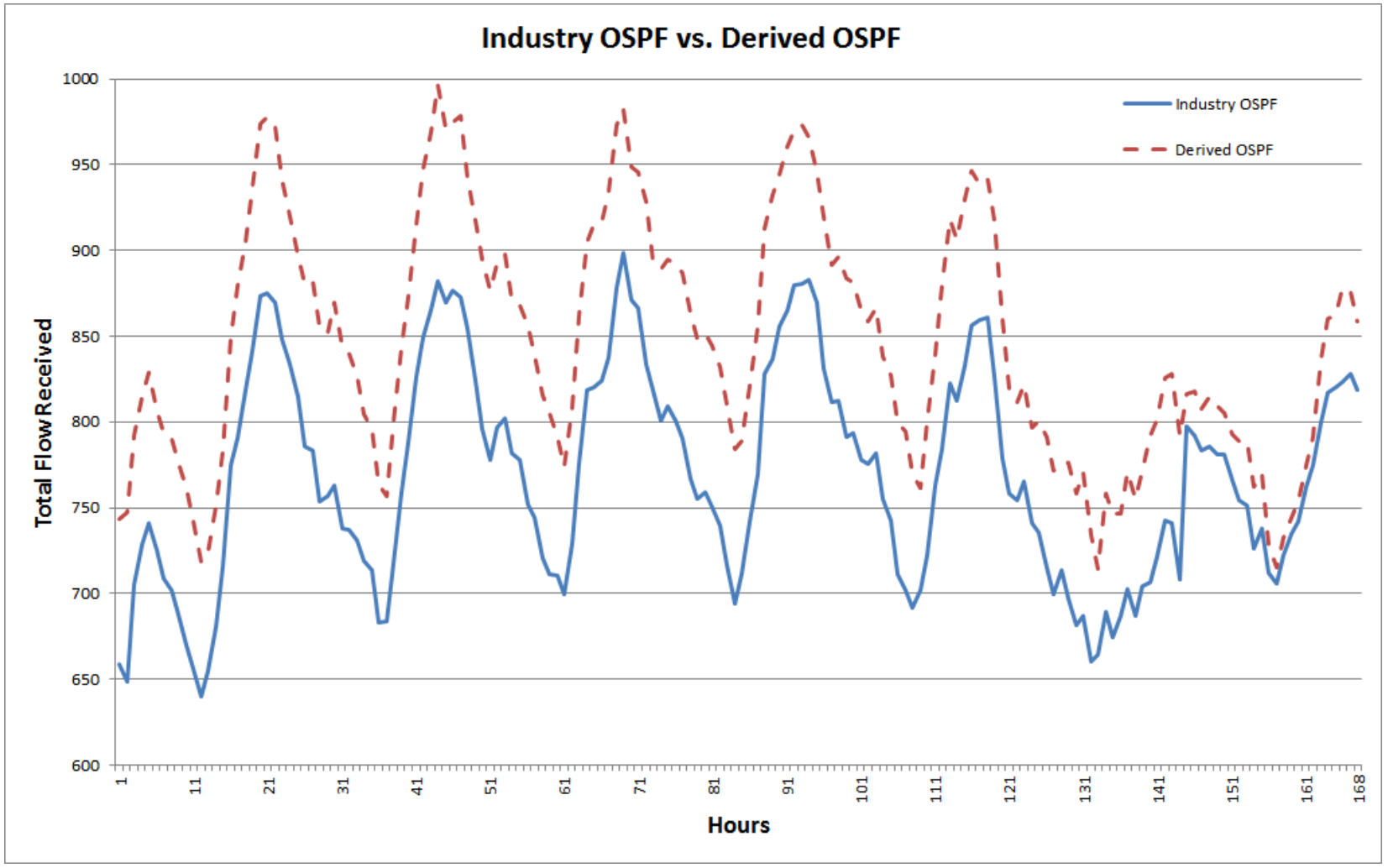

FIGURE 4.4: Industry OSPF and Derived OSPF for the Abilene Network

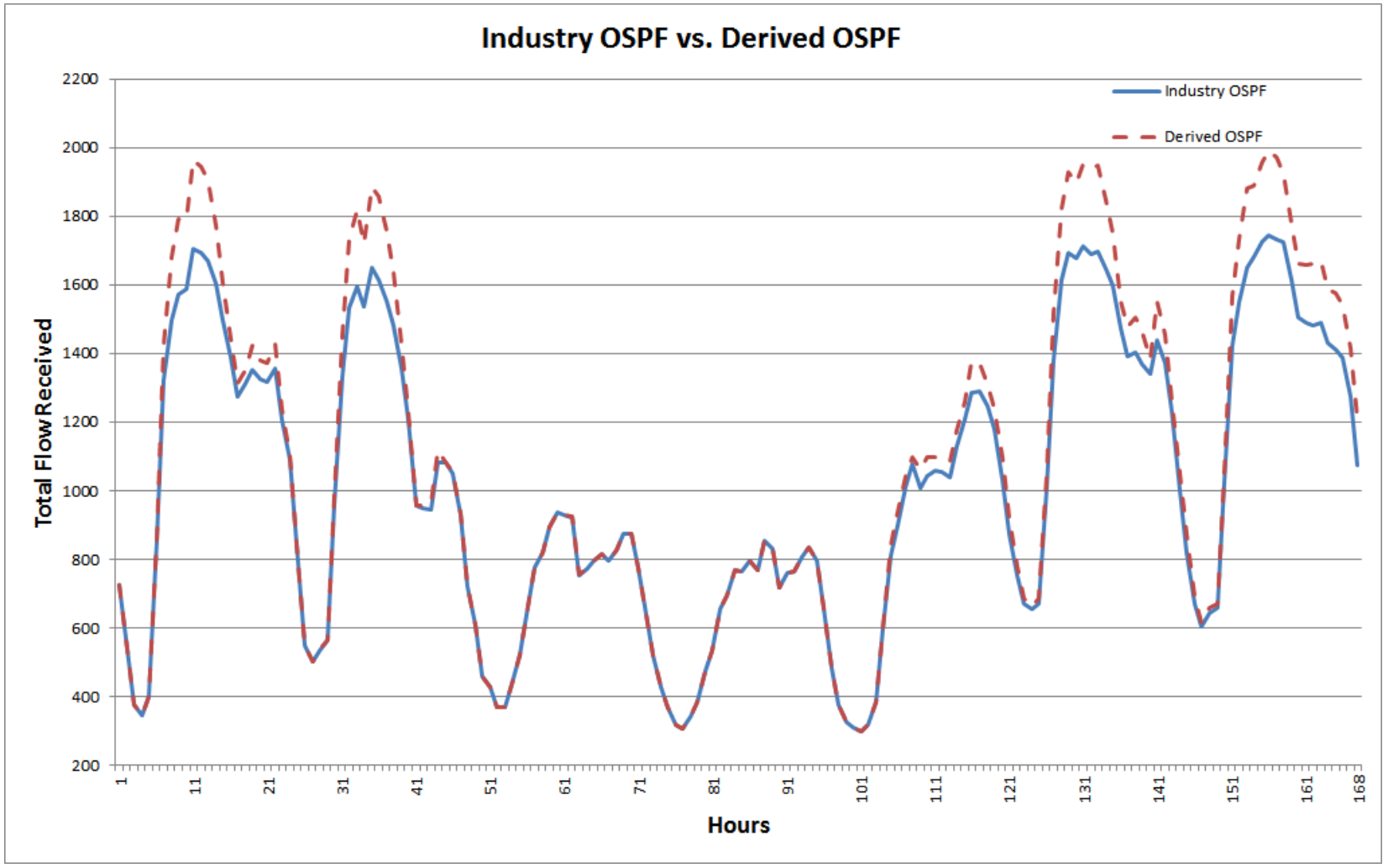

FIGURE 4.5: Industry OSPF and Derived OSPF for TWAREN

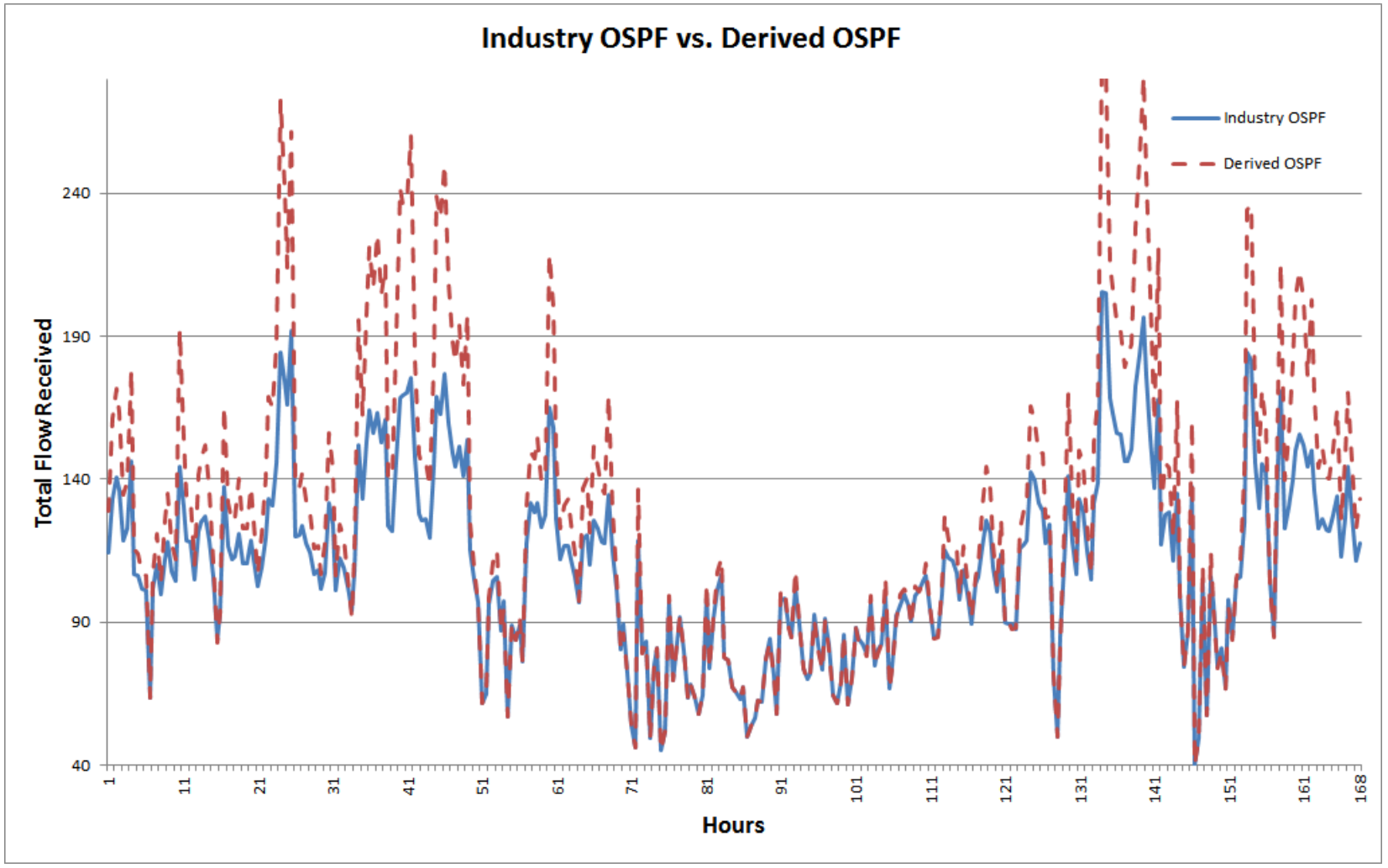

FIGURE 4.6: Industry OSPF and Derived OSPF for CANARIE

4.2.2 Robust Model

In practice, network arc weights are updated very infrequently. To find a good, long term policy, we formulate and solve the robust formulation of (4.4). The robust formulation maximizes the minimum weighted flow delivered across a set of realizations.

$$\max_{\alpha,\beta,x} \quad z \tag{4.5a}$$

$$z \leq \sum_{k=1}^{K} c^k x_{d^k}^{kq}, q \in Q \tag{4.5b}$$

$$\text{s.t.} \quad x_{o^k}^{kq} = s^{kq} \quad k \in T, q \in Q \tag{4.5c}$$

$$\sum_{j \in \delta^+(o^k)} \alpha_{o^k j}^k = 1 \quad k \in T \tag{4.5d}$$

$$\sum_{j \in \delta^-(d^k)} \alpha_{j d^k}^k = 1 \quad k \in T \tag{4.5e}$$

$$\sum_{j \in \delta^-(o^k)} \alpha_{j o^k}^k = 0 \quad k \in T \tag{4.5f}$$

$$\sum_{j \in \delta^+(d^k)} \alpha_{d^k j}^k = 0 \quad k \in T \tag{4.5g}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k = \sum_{i \in \delta^+(j)} \alpha_{ji}^k \quad k \in T, j \neq o^k, d^k \tag{4.5h}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k \leq 1 \quad k \in T, j \neq o^k, d^k \tag{4.5i}$$

$$\sum_{i \in \delta^-(j)} \alpha_{ij}^k x_i^{kq} f_{ij} (\sum_{l=1}^{K} \alpha_{ij}^l x_i^{lq}) - x_j^{kq} = 0 \quad k \in T, j \in N : j \neq o^k, q \in Q \tag{4.5j}$$

$$\alpha_{ij}^k \leq \beta_{ij}^u \quad u \in N, o^k = u \tag{4.5k}$$

$$\sum_{(i,j) \in A} \beta_{ij}^u = |N| - 1 \quad u \in N \tag{4.5l}$$

$$\alpha_{ij}^k \in \{0, 1\} \quad (i, j) \in A, k \in T \tag{4.5m}$$

$$\beta_{ij}^u \in \{0, 1\} \quad (i, j) \in A, u \in N \tag{4.5n}$$

, where Q represents the collection of traffic demands scenarios.

The following figure shows the comparison of three set of routing policies: industry benchmark, the policy found by (4.4) and the policy found by the robust model (4.5).

As expected, the robust model sacrifices slightly the objective value for practical significance, but on the other hand, it still outperforms the benchmark.

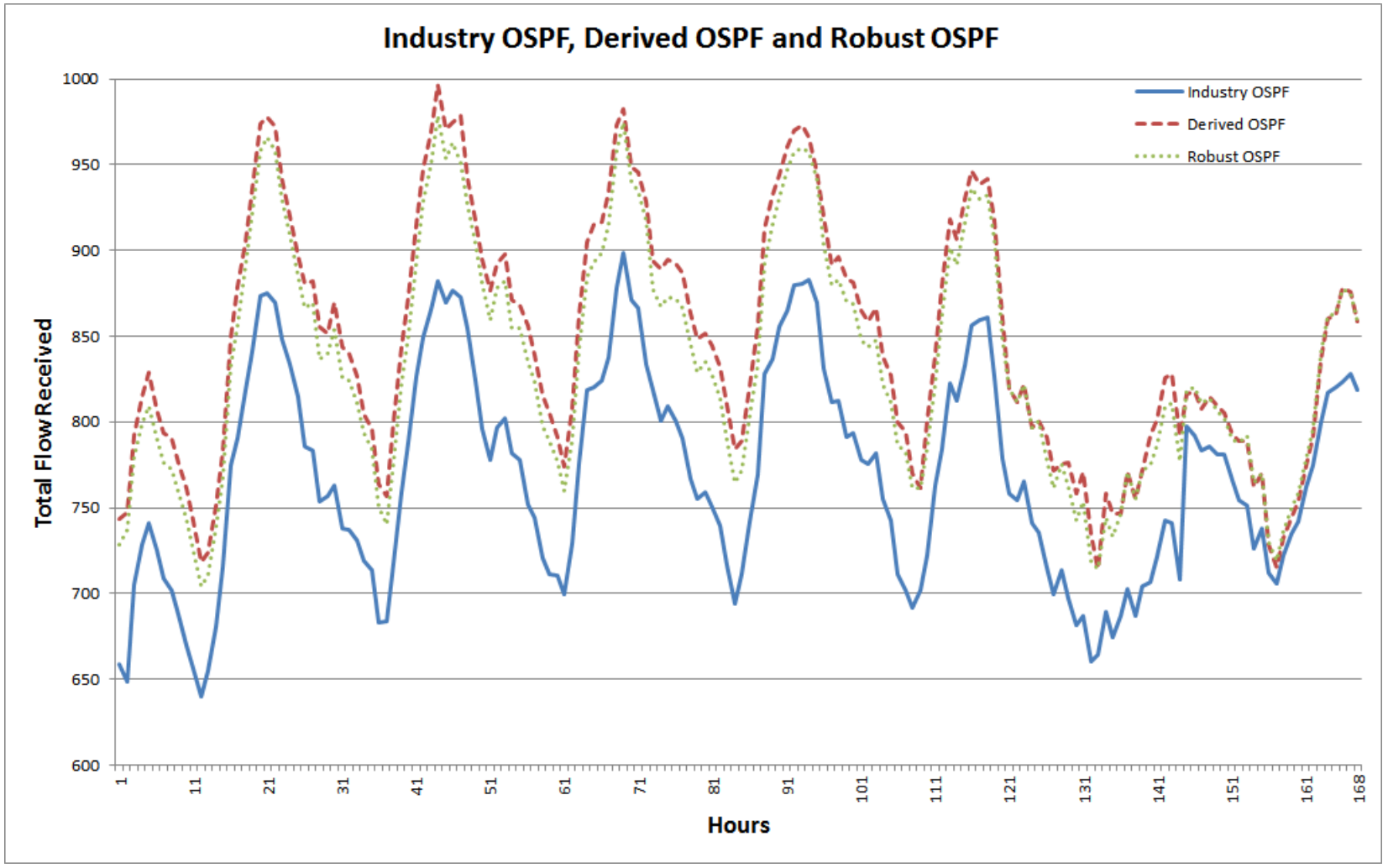

FIGURE 4.7: Industry OSPF, Derived OSPF and Robust OSPF for the Abilene Network

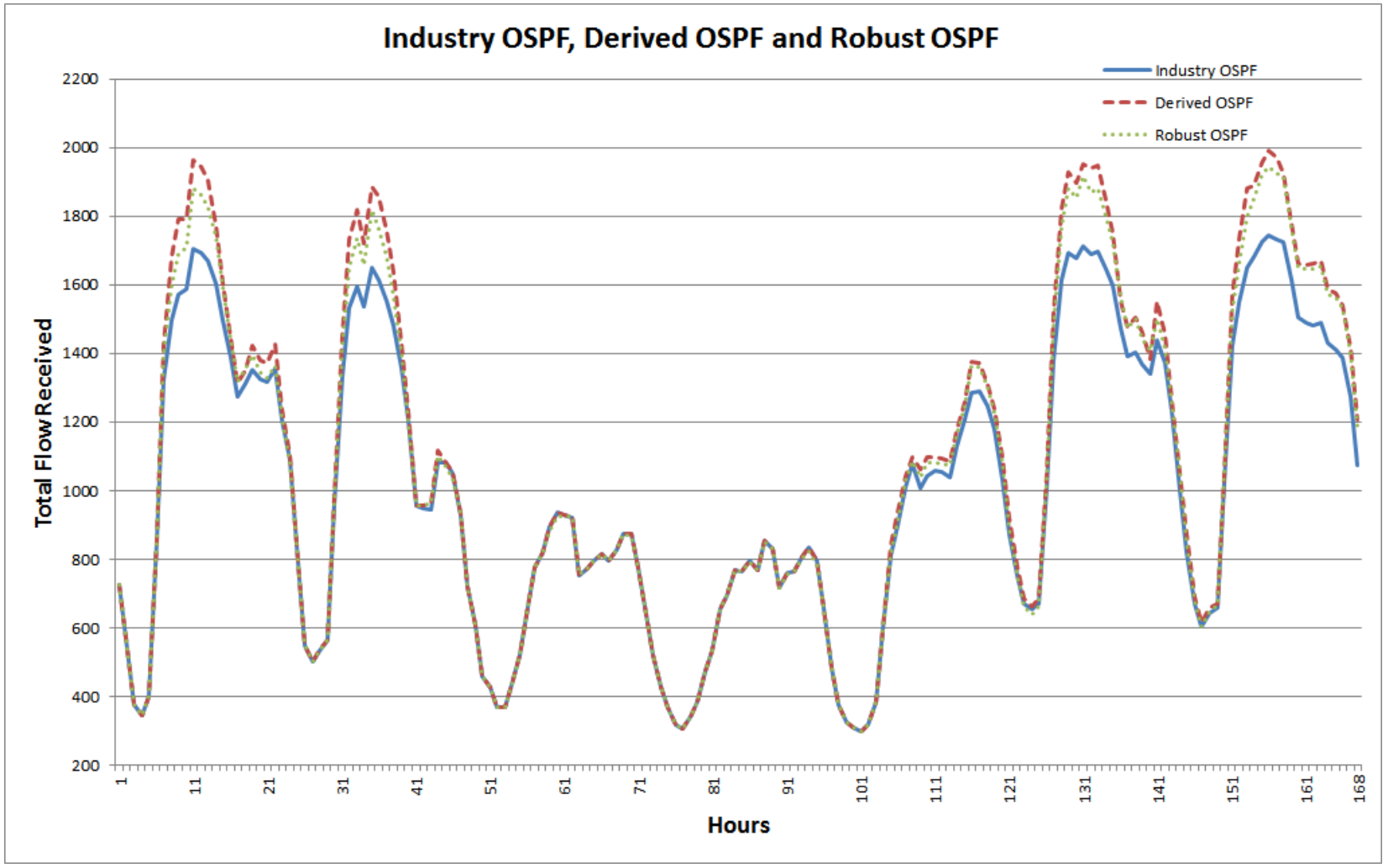

FIGURE 4.8: Industry OSPF, Derived OSPF and Robust OSPF for TWAREN

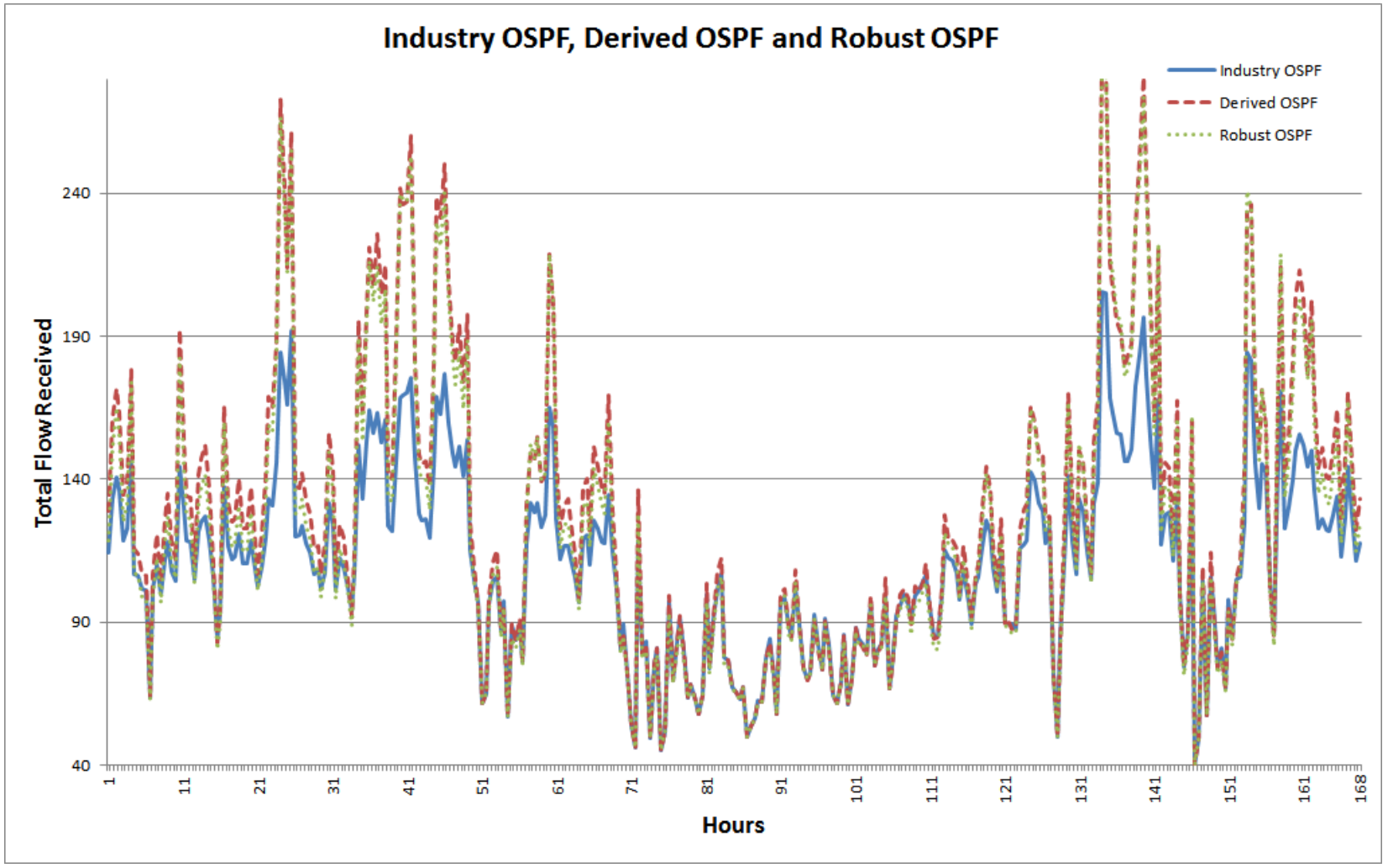

FIGURE 4.9: Industry OSPF, Derived and Robust OSPF OSPF for CANARIE

4.2.3 Analysis

For all three networks, the traffic demands over time all share a diurnal pattern as well as the weekday-weekend pattern. Specifically, the demands is higher in the daytime vesus in the night time, and higher during weekdays than weekends. For Abilene, the highest total demand in a week is 2 times the lowest total demand, while for TWAREN and CANARIE, the highest is about 4 times the lowest. Therefore, for Abilene, the total flow received is relatively stable, but TWAREN and CANARIE show a much greater variation in total flow received over a week. The demand variation reflects quite significantly in the comparison between the Industry policy and our policy, because suppose the demand drops below the minimum threshold, then under RED, every node will retain all the flow that is sent over, and therefore, the total flow received would be the same for any policy. Thus, as shown in Figure 4.10, 4.11, 4.12, 4.13, 4.14, 4.15,

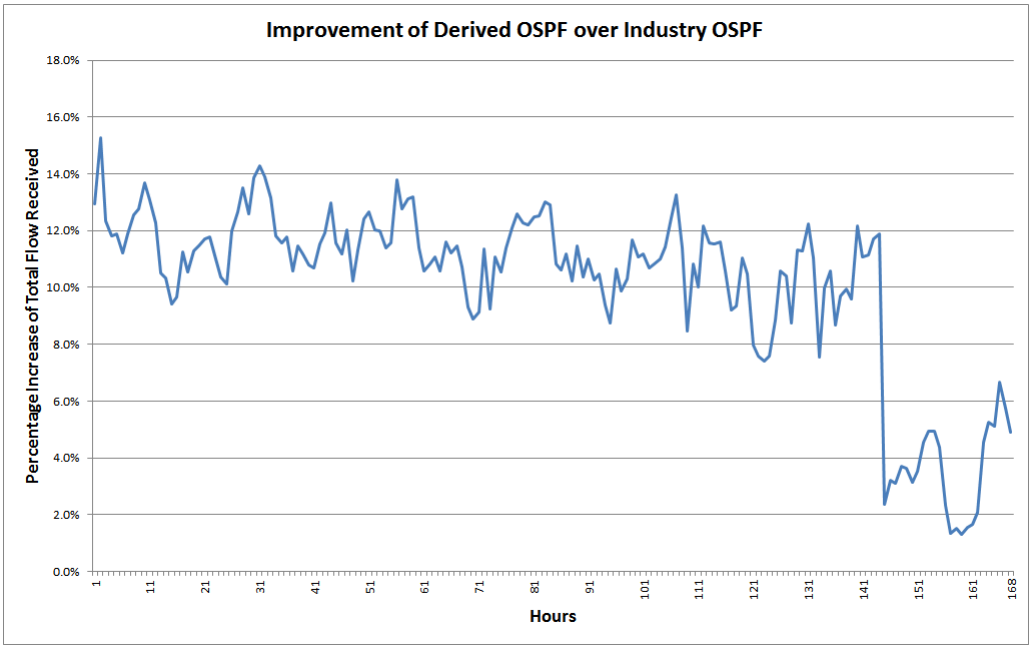

FIGURE 4.10: Percentage Improvement of derived policy over industry benchmark for Abilene Network

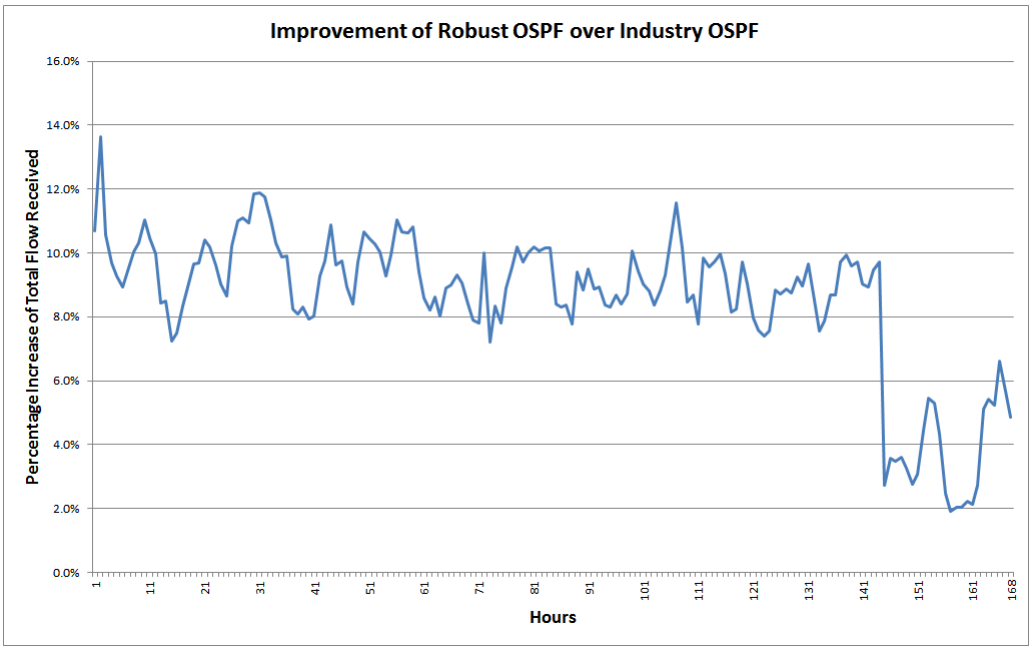

FIGURE 4.11: Percentage Improvement of robust policy over industry benchmark for Abilene Network

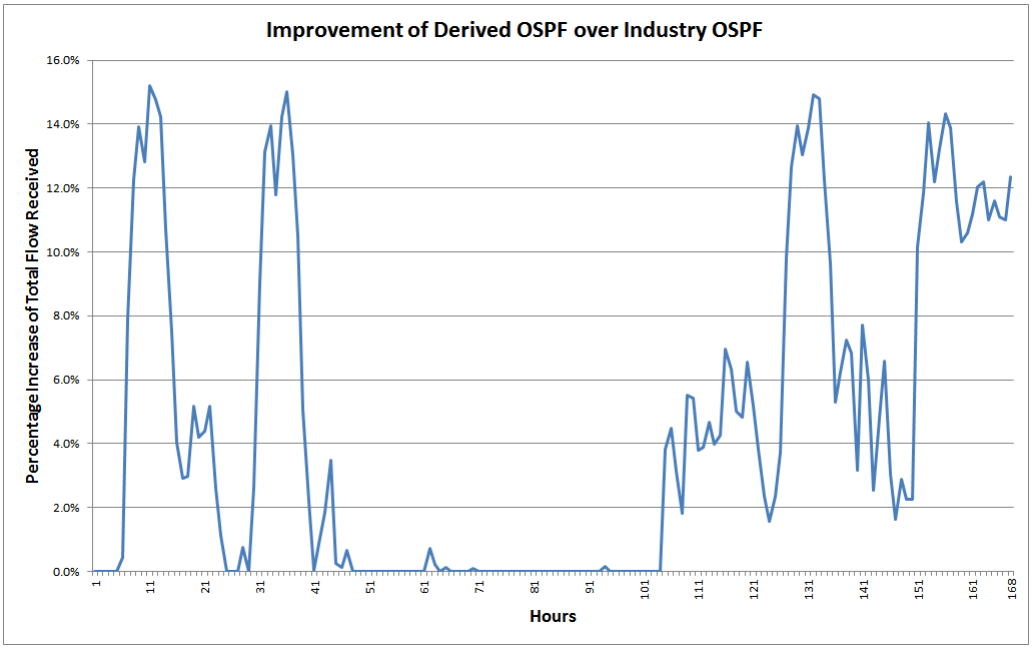

FIGURE 4.12: Percentage Improvement of derived policy over industry benchmark for TWAREN

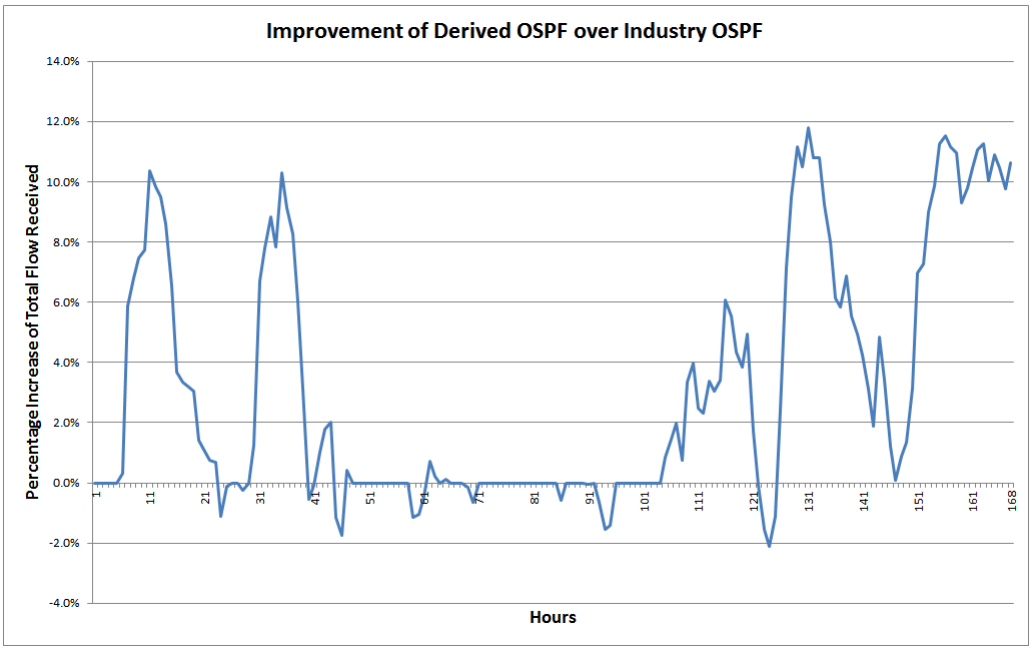

FIGURE 4.13: Percentage Improvement of robust policy over industry benchmark for TWAREN

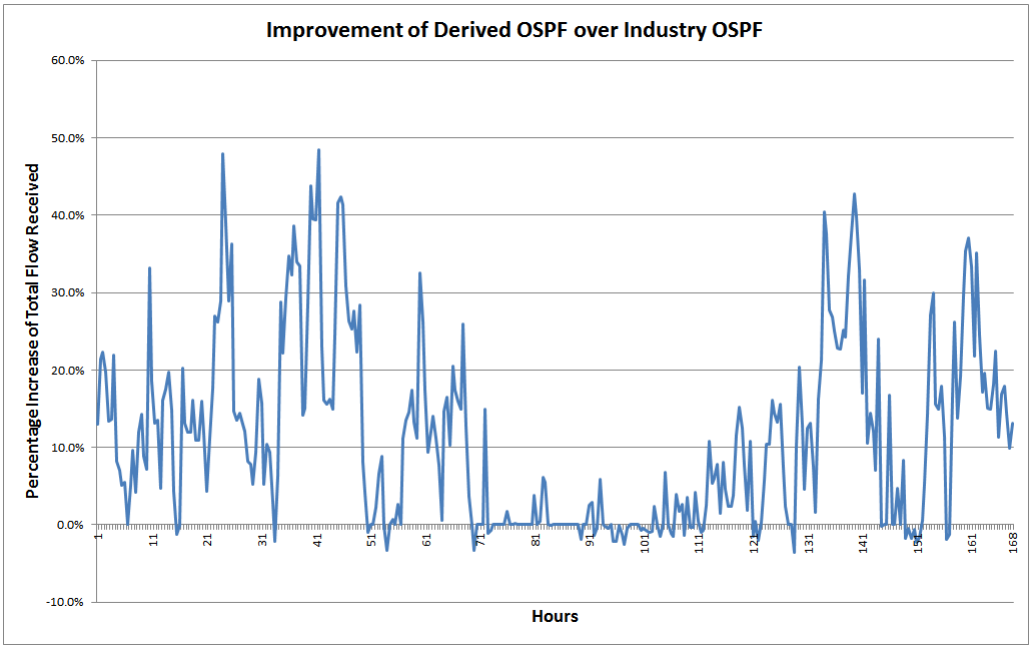

FIGURE 4.14: Percentage Improvement of derived policy over industry benchmark for CANARIE

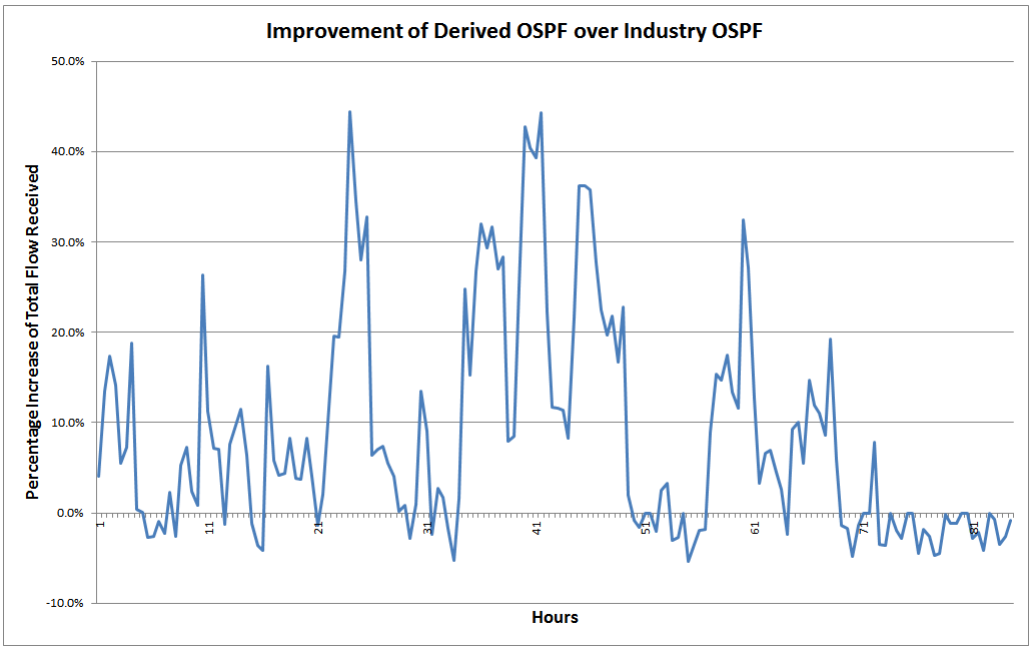

FIGURE 4.15: Percentage Improvement of robust policy over industry benchmark for CANARIE

for TWAREN and CANARIE, the percentage improvement of the total flow received from industry policy to our derived robust policy shows a similar diurnal pattern. For Abilene, the percentage increase is around 10%. Overall, these results show that our method outperforms the industry benchmark, for OSPF routing problem under RED.

4.2.4 Realizability

So far, we have obtained a feasible policy for each demand instance of Abilene, TWAREN and CANARIE. In order to make sure that these policies gain better performance without losing the OSPF compatibility, we need to solve model (3.5) to make sure we can find the corresponding OSPF arc weights for each policy. Recall that in (4.4), we introduce a tree constraint, a necessary condition of OSPF configurability. Our results show that every single one of the policy, regardless of whether it is a regular policy or robust policy, is OSPF-configurable, i.e., we obtain feasible arc metrics for each policy. This concludes the experiments.

4.3 Credit

The Abilene Network traffic demands used in our work were collected by Prof. Yin Zhang at University of Texas, Austin. The traffic data for CANARIE was provided by Thomas Lam and Jun Jian at CANARIE. We would like to thank them for providing all the real-world information for completing our work.

Chapter 5

Proof of Theorem 3.1

In this chapter, we present the proof of Theorem 3.1. Recall that the All-Pair Routing problem under RED is defined as follows:

Given: Let $G = (N, A)$ be a symmetrically directed graph, T be the set of all-pair commodities such that $|T| = |N| \cdot (|N| - 1)$. For each commodity $k \in T$, let c^k be the weight, $s^k > 0$ be the units of flow at the source node $o^k \in N$ to be sent to the destination node $d^k \in N$. Let f_{ij} be the RED function for each directed arc $(i, j) \in A$.

Find: A unique simple path $\{a^k\} \subseteq A$ for each $k \in T$.

Objective: Maximize the total weighted flow delivered under RED.

Recall Theorem 3.1:

Theorem. *All-Pair Routing Problem under RED is NP-Hard.*

We will use a reduction from the Set Cover problem (SCP) described as follows:

Given: A ground set $S = \{e_1, e_2, \ldots, e_n\}$ and a collection of subsets of S, $\mathcal{S} = \{S_1, \ldots, S_m\}$ s.t.

$$S = \bigcup_{j=1}^{m} S_j,$$

and a number k.

Find: a set cover of S, i.e., $J \subseteq \{1, 2, \ldots, m\}$ s.t.

$$S = \bigcup_{j \in J} S_j,$$

such that $|J| \leq k$.

SCP is NP-Hard [28]. This proof is inspired by Dimitrov's work [20].

5.1 Construction of the Instance

We construct an instance of All-Pair Routing problem under RED from a given SCP instance as follows: (an example is shown in Figure 5.1)

We construct a graph $G = (N, A)$ consisting of four node layers:

- Layer 1: n nodes $e_1, \ldots, e_n \in N$, each representing one element $e_i \in S$ in the SCP instance. We will use $\{e_i\}_{i=1}^n$ to represent Layer 1 nodes and elements in the SCP instance interchangeably.
- Layer 2: m nodes $S_1, \ldots, S_m \in N$, each representing one set S_j in the SCP instance. We will use $\{S_j\}_{j=1}^m$ to represent Layer 2 nodes and subsets in the SCP instance interchangeably.
- Layer 3: 1 node $I \in N$.
- Layer 4: 1 node $t \in N$.

Before we introduce the arc set A, we need to note that in order to define commodities for all pairs of nodes, the graph has to be undirected. However, to cope with the definition of the RED function that accounts for direction of arcs, we will use directed graph with symmetric arcs, that is, for each arc $(a, b) \in A$, there is an arc $(b, a) \in A$.

We add an arc pair $(e_i, S_j), (S_j, e_i) \in A$ if and only if $e_i \in S_j$ for some $e_i \in S$, $\forall j = 1, 2, \ldots, n$. In addition, we add an arc pair $(S_j, I), (I, S_j) \in A$, $\forall j = 1, 2, \ldots, m$, and finally we add an arc pair $(I, t), (t, I) \in A$.

The all-pairs commodities are defined as follows:

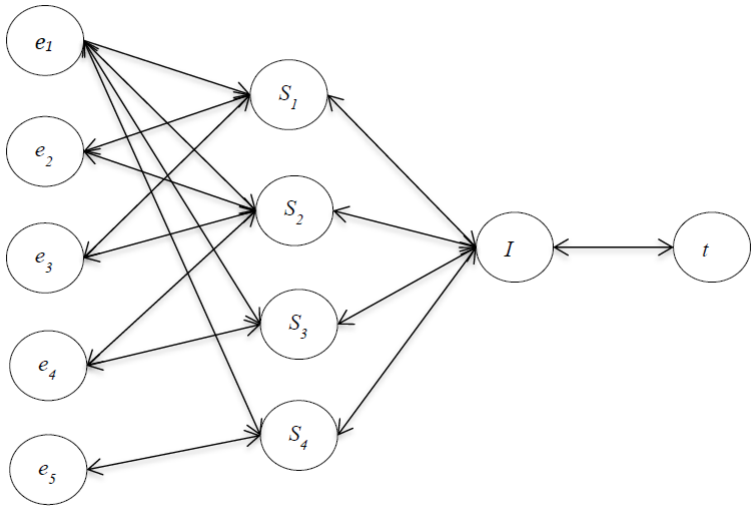

FIGURE 5.1: An example

- Type I (T_1): $s^k = 1$, $o^k = e_i, i = 1, 2, \ldots, n$, $d^k = t$ and $c^k = 0$, $\forall k \in T_1$. There are n Type I commodities.

- Type II (T_2): $s^k = 1$, $o^k = S_j$, $d^k = e_i$, $\forall (e_i, S_j) \in A$ and $c^k = L = |T_2| + n + 4$, $\forall k \in T_2$. There are $|A| - m - 1$ Type II commodities.

- Type III (T_3): $s^k = 1$, $o^k = I$, $d^k = t$ and $c^k = 1$, $\forall k \in T_3$. There is 1 Type III commodity.

- Type IV (T_4): $0 < \epsilon < \frac{1}{\eta\Delta}$ unit between all the other pairs of nodes, with weight $c^k = 0, \forall k \in T_4$, where $\eta = (n + m + 2)(n + m + 1) - n - (|A| - m - 1) - 1$ is the number of Type IV commodities, let $R = \eta\epsilon < \frac{1}{\Delta}$ denote the total units of residuals. Δ is a real number large enough such that $\Delta > \max\{2, (|T_2| + n + 4)^2, |T_2| + n + 1, \frac{1}{\sqrt{\frac{m+1}{m}} - 1}\}$, to be used in the proof of Claim 5.7 and Claim 5.8. Since the amount of Type IV commodities are very insignificant in the network, we will call it Type IV residuals, or residuals for the rest of the proof.

Let $T = T_1 \cup T_2 \cup T_3 \cup T_4$.

Let x_v^k be the amount of flow received at node v for commodity k, and by assumption $x_{o^k}^k = s^k$. Note that with the weight $c^k, k \in T$ defined above, the objective of this All-Pair Routing Problem under RED instance is to maximize the following expression:

$$\sum_{k \in T_2} L \cdot x_{e_{i_k}}^k + x_t^{k^3},$$

where $T_3 = \{k^3\}$, and i_k is the destination of the Type II commodity k.

We define our RED function as follows: for each arc (u, v)

$$f_{uv}(t) = \begin{cases} 1 & \text{if } t \leq \kappa_{uv} \\ \frac{\kappa_{uv}}{t} & \text{o.w.} \end{cases},$$

where

$$\kappa_{uv} = \begin{cases} 1 + R & \text{if } (u, v) = (e_i, S_j) \in A \text{ or } (u, v) = (S_j, e_i) \in A \\ 0.5 & \text{if } (u, v) = (I, S_j), j = 1, 2, \ldots, m \\ 1 & \text{o.w.} \end{cases}.$$

With the RED function defined above, we have the following claim:

Claim 5.1. *Consider an arc (i, j) and its corresponding RED function f_{ij}. Suppose the commodities $k_1, k_2, \ldots, k_s$ sent over this arc have $t_1, t_2, \ldots, t_s$ units, respectively, and let $\bar{t} = \sum_{i=1}^{s} t_i$, then the amount of commodity k_l received at node j is*

$$x_j^{k_l} = \begin{cases} t_l & \text{if } \bar{t} < \kappa_{ij} \\ t_l \frac{\kappa_{ij}}{\bar{t}} & \text{if } \bar{t} \geq \kappa_{ij} \end{cases},$$

for $l = 1, 2, \ldots, s$. Moreover, the total flow received at j is equal to $\min\{\bar{t}, \kappa_{ij}\}$.

Proof. From the definition of the RED function f_{ij}, we know that the amount of k_l received at j is the flow sent from i, $x_i^{k_l} = t_l$ multiplied by the RED discount $f_{ij}(t)$:

$$x_j^{k_l} = t_l \cdot f_{ij}(t) = \begin{cases} t_l & \text{if } \bar{t} < \kappa_{ij} \\ t_l \frac{\kappa_{ij}}{\bar{t}} & \text{if } \bar{t} \geq \kappa_{ij} \end{cases},$$

The total flow received is

$$\begin{aligned} \sum_{l=1}^{s} x_j^l &= \begin{cases} \sum_{l=1}^{s} t_l = \bar{t} & \text{if } \bar{t} < \kappa_{ij} \\ \sum_{l=1}^{s} t_l \frac{\kappa_{ij}}{\bar{t}} = \kappa_{ij} & \text{if } \bar{t} \geq \kappa_{ij} \end{cases}, \\ &= \min\{\bar{t}, \kappa_{ij}\} \end{aligned}$$

as required.

□

Remark 5.2. For any arc $(i, j) \in A$, if the total flow over this arc is no more than the threshold κ_{ij}, then all the flow is preserved, otherwise, all the flow will be proportionally reduced and the sum of the total flow retained is equal to the threshold κ_{ij}. An example is shown in Figure 5.2.

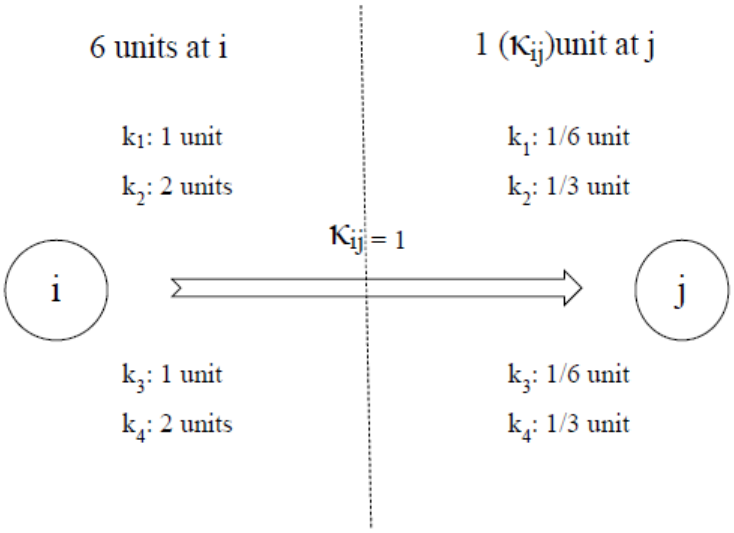

FIGURE 5.2: Proportional discount

We will now prove that there exists a set cover of size at most Q if and only if there exists a solution for the All-Pair Routing problem under RED instance of value at least $L \cdot |T_2| + \frac{1}{1+Q\frac{\Delta+1}{\Delta}}$.

5.2 A "Great" Solution

In this section, we will construct a feasible solution, namely a "great" solution, and discuss its properties.

Consider a feasible solution as follows. Type I commodities are sent over path $e_i \to S_j \to I \to t$, for some S_j. Type II commodities are sent over the single arc (S_j, e_i) that connects its source and its destination. Type III commodities are sent over the single arc (I, t), and residuals are sent freely. As, residuals are set freely, the "great" solution,

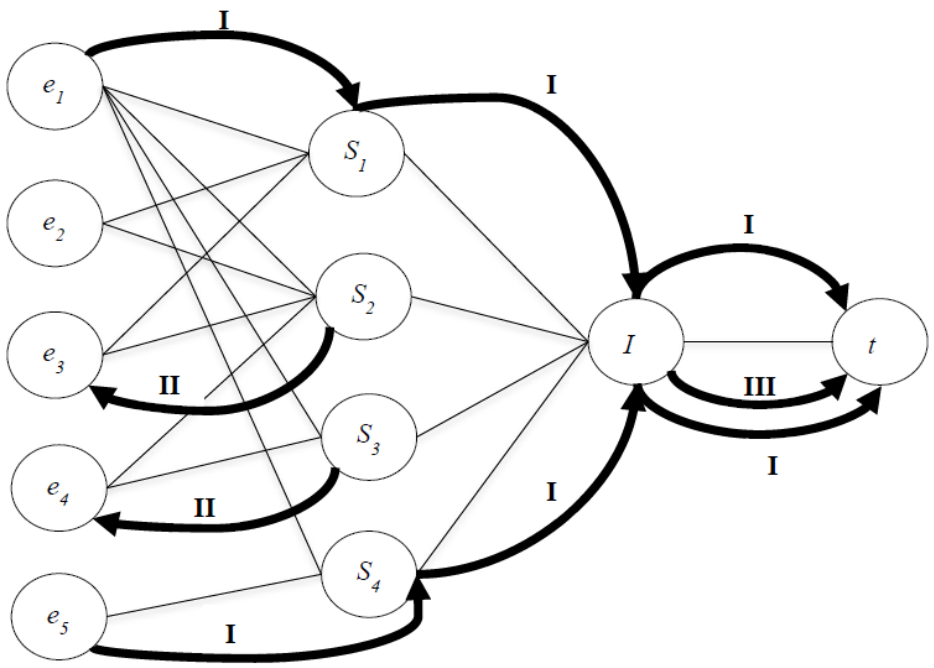

FIGURE 5.3: A "great" solution

is in fact, a class of solutions, and not a single solution, but for easy of exposition we treat this class as a singleton because residuals do not impact our reduction.

With this solution described above, we are able to calculate the flow for each commodity as well as the objective value. Note that only the Type II and III commodities have positive weights. Consider a Type II commodity from S_j to e_i, for some $(e_i, S_j) \in A$. This commodity is sent through arc (S_j, e_i), and all other commodities in T that possibly use the same arc are the residuals, as indicated in Figure 5.3. Note that there are at most R units of residuals over this arc. As a result, the total flow over this arc is at most $1 + R = \kappa_{S_j e_i}$. By Claim 5.1 and Remark 5.2, all 1 unit of this Type II commodity is preserved at its destination e_i. Therefore, the total weighted flow received for all Type II commodities is

$$\sum_{k \in T_2} L \cdot 1 = L \cdot |T_2|.$$

For Type I commodities, consider a Layer 2 node S_j. Suppose we send the Type I commodity starting from e_i over the path $e_i \to S_j \to I \to t$ by definition. Since no

other Type I commodity is sent over the arc (e_i, S_j), the total flow sent over this arc is at most $1+R$ when accounting for residuals, and since the threshold $\kappa_{e_i S_j} = 1+R$, the total flow does not exceed the threshold, therefore Claim 5.1 implies that all one unit of this Type I commodity starting from e_i will be preserved at S_j. Let t_j be the units of Type I commodities received at S_j, then t_j counts the number of Type I commodities that is sent to S_j, i.e.,

$$t_j = |\{k \in T_1 : k \text{ is sent to } S_j\}| \in \mathbb{N}.$$

Note that $t_j \geq 1$ whenever $t_j > 0$. Let $J = \{j : t_j > 0\}$ denote the collection of sets S_j that carries Layer I commodities.

When the Type I commodities are sent from Layer 2 to I, since $\kappa_{S_j I} = 1. \forall j = 1, 2, \ldots, m$, we know that $t_j \geq \kappa_{S_j I}, \forall j \in J$. Therefore by Claim 5.1, the total amount of flow will be

$$\begin{cases} \frac{t_j}{t_j + r_j} \in \left(\frac{t_j}{t_j+R}, 1\right) & j \in J \\ 0 & j \notin J \end{cases},$$

where $r_j \in (0, R)$ is the amount of residuals over arc (S_j, I), for each $j \in J$. Therefore, the total flow of Type I commodity received at I is simply the sum over J, and is within the range

$$\left(\sum_{j \in J} \frac{t_j}{t_j + R}, |J|\right).$$

Finally, all the Type I commodities will be sent from I to t and at the same time the Type III commodity will also be sent from I to t, and Claim 5.1 implies that the amount of flow of Type III commodity k^3 received at t is $x_t^{k^3} = \frac{1}{1+R_1+R_2}$ where $R_1 \in \left(\sum_{j \in J} \frac{t_j}{t_j+R}, |J|\right)$ is the amount of Type I commodities sent over (I, t) and $R_2 \in (0, R)$ is the amount of residuals sent over (I, t). In summary, we have the following claim:

Claim 5.3. *For any "great" solution, the objective value is*

$$L \cdot |T_2| + \frac{1}{1 + R_1 + R_2},$$

where $R_1 \in \left(\sum_{j \in J} \frac{t_j}{t_j+R}, |J|\right)$ *and* $R_2 \in (0, R)$.

Proof. Note that the objective value by definition is

$$\sum_{k \in T_2} L \cdot x^k_{e_{i_k}} + x^{k^3}_t,$$

and $x^k_{e_{i_k}} = 1$, $\forall k \in T_2$ and $x^{k^3}_t = \frac{1}{1+R_1+R_2}$, we know that the objective value for a "great" solution is:

$$L \cdot |T_2| + \frac{1}{1 + R_1 + R_2},$$

where $R_1 \in \left(\sum_{j \in J_1} \frac{t_j}{t_j + R}, |J|\right)$ is the amount of Type I commodities sent over (I, t) and $R_2 \in (0, R)$ is the amount of residuals sent over (I, t). □

Moreover, we have the following claim:

Claim 5.4. *For any "great" solution, let J be the collection of subset S_j where S_j, as a Layer 2 node, carries Type I flow. Then J is a set cover for the original SCP instance.*

Proof. A Type I commodity is uniquely determined by its source node e_i and is sent over an arc (e_i, S_j) if $e_i \in S_j$ in the original set cover instance. By definition $J = \{j : t_j > 0\}$ represents all Layer 2 nodes that carry some Type I commodities. Consider an element $e_i \in S$, and the corresponding Type I commodity determined by e_i. We can find a node $S_j \in J$ that carries this commodity sent from e_i, thus $(e_i, S_j) \in A$, or equivalently, $e_i \in S_j$. Therefore, $e_i \in \bigcup_{S \in J} S$ and J is a set cover. □

We call a set cover induced by a "great" solution, if it is defined as in Claim 5.4.

5.3 Proof: Part I

In this section, we will show that if there exists a set cover J_1 of size Q, then there exists a solution to our problem of value at least $L \cdot |T_2| + \frac{1}{1+Q\frac{\Delta+1}{\Delta}}$.

We construct a "great" solution as defined in Section 5.2. Specifically, for Type I commodities that starts from e_i, we choose $S_j \in J_1$ that contains e_i. Note that Type II, III, IV commodities are defined uniquely for any "great" solution. Then by Claim 5.3 the objective value of this solution is

$$
\begin{aligned}
L \cdot |T_2| + \frac{1}{1 + R_1 + R_2} &\geq L \cdot |T_2| + \frac{1}{1 + |J_1| + R} \\
&> L \cdot |T_2| + \frac{1}{1 + |J_1| \frac{\Delta+1}{\Delta}} \ , \\
&= L \cdot |T_2| + \frac{1}{1 + Q \frac{\Delta+1}{\Delta}}
\end{aligned}
$$

where $R_1 \in \left(\sum_{j \in J} \frac{t_j}{t_j + R}, |J| \right)$ and $R_2 \in (0, R)$, and $R < \frac{1}{\Delta} \leq \frac{|J_1|}{\Delta}$ by definition.

Therefore, the objective value of this solution is at least $L \cdot |T_2| + \frac{1}{1+Q\frac{\Delta+1}{\Delta}}$.

5.4 Proof: Part II

In this section, we will show that if we can find a solution to our problem of value at least $L \cdot |T_2| + \frac{1}{1+Q\frac{\Delta+1}{\Delta}}$, then the original SCP instance has a set cover of size Q.

We call a solution of value at least equal to $L \cdot |T_2| + \frac{1}{1+Q\frac{\Delta+1}{\Delta}}$ a "good" solution. We will show that a "good" solution is indeed a "great" solution.

Claim 5.5. *For any "good" solution, the Type II commodities never reach I.*

Proof. Consider a feasible solution where there is a Type II commodity k that reaches I. Since this commodity starts at a Layer 2 node S_{j_1}, so when it reaches I, it must be sent back to some other Layer 2 node S_{j_2}. The RED threshold is $k_{IS_{j_2}} = 0.5$, so even if this commodity is the only one that is sent through this arc, the amount received at S_{j_2} is 0.5, as shown in Figure 5.4. Clearly if there are other commodities sharing the arc (I, S_{j_2}), commodity k received at S_{j_2} will be reduced even more. Thus, assuming all other commodities are full preserved at their destination, we can calculate the the upper bound of the objective value:

$$
\begin{aligned}
\underbrace{L(|T_2| - 1)}_{T_2 \setminus \{k\}} + \underbrace{L \cdot 0.5}_{k} + \underbrace{1}_{T_3} &= L(|T_2| - 0.5) + 1 \\
&= L \cdot |T_2| - (L \cdot 0.5 - 1) \\
&< L \cdot |T_2| \\
&< L \cdot |T_2| + \frac{1}{1 + Q\frac{\Delta+1}{\Delta}}
\end{aligned}
,
$$

since $L = |T_2| + n + 4 > 2$.

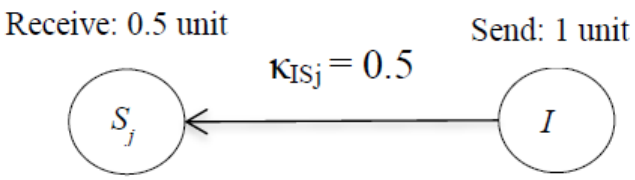

FIGURE 5.4: Discount of a Type II commodity that reaches I

The objective value of this solution is strictly less than the objective value of a "good" solution, a contradiction. Therefore for any "good" solution, the Type II commodities never reach I.

□

Claim 5.6. *For any "good" solution, the Type II commodities always use the single arc (S_j, e_i) only.*

Proof. From Claim 5.5, we know that in a "good" solution, Type II commodities will never reach I. Consider a feasible solution where Type II commodities do not reach I and there is one Type II commodity k_1 from S_1 to e_2 that does not use the single arc (S_1, e_2) only, implying that this commodity must use at least two arcs of type (S_j, e_i). Since there is one commodity for each arc connecting Layer 2 and Layer 1 nodes, there must exist some arc (S_2, e_2) that carries k_1 and another Type II commodity k_2, as illustrated in Figure 5.5. In this case, the total flow received at e_2 will be at most $\kappa_{S_2 e_2} = 1 + R$, meaning that the total flow for these two commodities received at destinations is bounded above by $1 + R$. Thus, assuming all other commodities are fully preserved at destinations, the value of this solution is at most

$$\begin{aligned}
\underbrace{L(|T_2| - 2)}_{T_2 \setminus \{k_1, k_2\}} + \underbrace{L \cdot (1 + R)}_{k_1, k_2} + \underbrace{1}_{T_3} &= L(|T_2| - 1 + R) + 1 \\
&\leq L \cdot |T_2| - \frac{1}{2} L + 1 \\
&< L \cdot |T_2| \\
&< L \cdot |T_2| + \frac{1}{1 + Q\frac{\Delta+1}{\Delta}}
\end{aligned},$$

where we use the fact that $R \leq \frac{1}{\Delta} \leq \frac{1}{2}$ and $L = |T_2| + n + 4 > 2$.

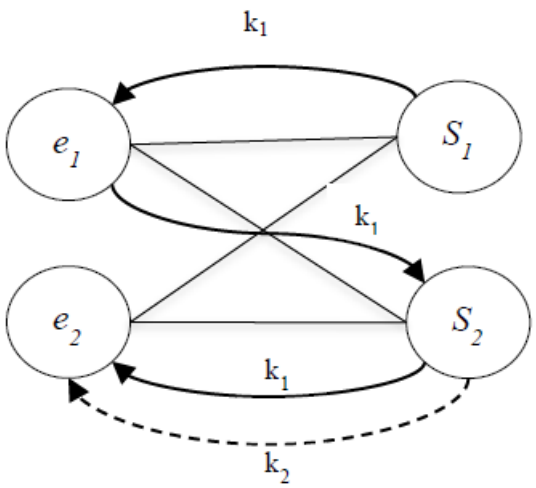

FIGURE 5.5: $k_1 : S_1 \to e_1 \to S_2 \to e_2$; $k_2 : S_2 \to e_2$, sharing (S_2, e_2)

The objective value of this solution is strictly less than the objective value of a "good" solution, a contradiction. Therefore, for any "good" solution, Type II commodities always use the single arc (S_j, e_i) only.

□

Claim 5.7. *For any "good" solution, the Type I commodities will never reach a Layer 1 node other than its source.*

Proof. Consider a "good" solution where all the Type II commodities are sent through the single arc (S_j, e_i), due to Claim 5.6. First note that each commodity is sent through a simple path, which is a node-disjoint path that does not contain cycle; thus, whenever a Type I commodity reaches I, it will be directed forwarded to t, otherwise it has to be back at I again in order to reach t, creating a cycle. Let k_1 be a Type I commodity starting from e_1, and it reaches S_1 after first step. The amount of k_1 received at S_1, $x_{S_1}^{k_1}$ is bounded above by 1, when it is fully preserved. A lower bound of $x_{S_1}^{k_1}$, on the other hand, is obtained when all commodities in T are sent over (e_1, S_1). In this case, as $\kappa_{e_1,S_1} = 1 + R$, the amount of k_1 received at S_1 is

$$\begin{aligned}\frac{1}{|T_1 \cup T_2 \cup T_3| + R} \cdot (1 + R) &= \frac{1}{|T_2| + n + 1 + R} \cdot (1 + R) \\ &\geq \frac{1}{|T_2| + n + 1} \\ &\geq \frac{1}{\Delta} > R\end{aligned} .$$

Suppose k_1 goes back from S_1 to Layer 1 at e_2. k_1 must share the arc (S_1, e_2) with the corresponding Type II commodity k_2, as shown in Figure 5.6. Since the amount of k_2 at S_2, $x_{S_2}^{k_2} = 1$, the total flow sent over arc (S_1, e_2) is at least $1 + \frac{1}{|T_2|+n+1+R} > 1 + R = \kappa_{S_1 e_2}$. Therefore, the flow will be reduced and the smaller $x_{S_1}^{k_1}$ is, the greater amount of k_2 is received at the destination e_2. Consider the largest possible amount of k_2 received at e_2, $x_{e_2}^{k_2}$. We want to set $x_{S_1}^{k_1} = \frac{1}{|T_2|+n+1+R} \cdot (1 + R)$, the lower bound of $x_{S_1}^{k_1}$. In addition, none of the other commodities is sent over (S_1, e_2) to minimize the load of this arc, then the amount of k_2 received at the destination e_1 is

$$\frac{1+R}{1 + \frac{1}{|T_1 \cap T_2 \cap T_3|+R} \cdot (1+R)} < 1.$$

Therefore, assuming all commodities other than k_1, k_2 are fully preserved, the objective value is

$$\begin{aligned}
&\underbrace{L(|T_2|-1)}_{T_2 \setminus \{k_2\}} + \underbrace{L \cdot \frac{1+R}{1 + \frac{1}{|T_2|+n+1+R} \cdot (1+R)}}_{k_2} + \underbrace{1}_{T_3} \\
=&L \cdot |T_2| - (L\left(1 - \frac{1+R}{1 + \frac{1}{|T_2|+n+1+R} \cdot (1+R)}\right) - 1) \\
=&L \cdot |T_2| - \left(L \frac{\frac{1}{|T_2|+n+1+R} \cdot (1+R) - R}{1 + \frac{1}{|T_2|+n+1+R} \cdot (1+R)} - 1\right) \\
=&L \cdot |T_2| - \left(\frac{L(1+R) - LR(|T_2|+n+1+R)}{|T_2|+n+1+R+1+R} - 1\right) \\
<&L \cdot |T_2| - \left(\frac{L(1 - \frac{1}{\Delta}(|T_2|+n+R))}{|T_2|+n+2+2R} - 1\right) \\
<&L \cdot |T_2| - \left(\frac{L(1 - \frac{|T_2|+n+R}{(|T_2|+n+4)^2})}{|T_2|+n+2+2R} - 1\right) \\
<&L \cdot |T_2| - \left(\frac{L(1 - \frac{1}{|T_2|+n+4})}{|T_2|+n+2+2R} - 1\right) \\
=&L \cdot |T_2| - \left(\frac{L - \frac{|T_2|+n+4}{|T_2|+n+4}}{|T_2|+n+2+2R} - 1\right) \\
<&L \cdot |T_2| - \left(\frac{L-1}{|T_2|+n+2+2R} - 1\right) \\
<&L \cdot |T_2| - \left(\frac{|T_2|+n+3}{|T_2|+n+2+2R} - 1\right) \\
<&L \cdot |T_2| - (1-1) \\
<&L \cdot |T_2| + \frac{1}{1 + Q\frac{\Delta+1}{\Delta}}
\end{aligned},$$

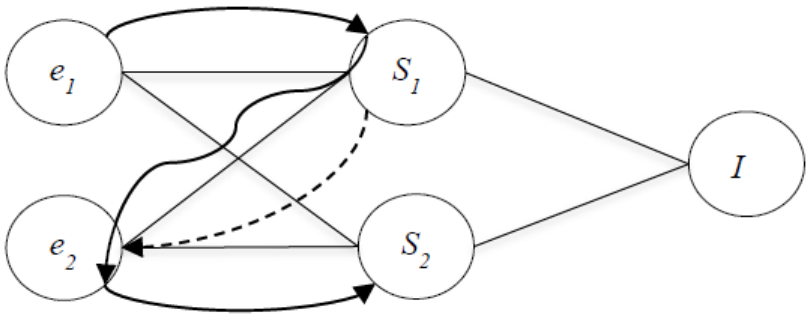

FIGURE 5.6: Full line: Type I; dotted: Type II, sharing (S_1, e_2)

since $L = |T_2| + n + 4$ by definition and $R < \frac{1}{\Delta} < \min\left(\frac{1}{2}, \frac{1}{(|T_2|+n+4)^2}\right)$.

The objective value of this solution is strictly less than the objective value of a "good" solution, a contradiction. Therefore, for any "good" solution, Type I commodities never reach a Layer 1 node other than its source.

□

So far, combining Claim 5.4,5.5,5.6, we realize that a "good" solution is indeed a "great" solution, defined in Section 5.2. Therefore, we know that the "good" solution objective value equals to

$$L \cdot |T_2| + \frac{1}{1 + R_1 + R_2},$$

where where $R_1 \in \left(\sum_{j \in J_2} \frac{t_j}{t_j + R}, |J_2|\right)$ is the amount of Type I commodities sent over (I, t) and $R_2 \in (0, R)$ is the amount of residual commodity sent over (I, t), J_2 is the set cover induced by this "good" solution. Finally, we will claim that J_2 is the set cover that we are searching for at the beginning of this section.

Claim 5.8. $|J_2| \leq Q$.

Proof. Assume otherwise that $|J_2| > Q$. Since any "good" solution is a great solution, by Claim 5.3, the objective value of this solution is

$$L \cdot |T_2| + \frac{1}{1 + R_1 + R_2},$$

where $R_1 \in \left(\sum_{j\in J} \frac{t_j}{t_j+R}, |J|\right)$ and $R_2 \in (0, R)$. Therefore we have

$$\begin{aligned}
\frac{1}{1+R_1+R_2} &\leq \frac{1}{1+\sum_{j\in J_2} \frac{t_j}{t_j+R}} \\
&< \frac{1}{1+\sum_{j\in J_2} \frac{t_j}{t_j+\frac{1}{\Delta}}} \\
&\leq \frac{1}{1+\sum_{j\in J_2} \frac{t_j}{t_j+\frac{t_j}{\Delta}}}, \\
&< \frac{1}{1+\sum_{j\in J_2} \frac{\Delta}{\Delta+1}} \\
&< \frac{1}{1+|J_2|\frac{\Delta}{\Delta+1}} \\
&< \frac{1}{1+Q\frac{\Delta+1}{\Delta}}
\end{aligned} \tag{5.1}$$

where the last step of (5.1) follows due to the fact that $\Delta > \frac{1}{\sqrt{\frac{m+1}{m}}-1} \Rightarrow \frac{(\Delta+1)^2}{\Delta^2} < \frac{m+1}{m}$ and thus we have

$$\begin{aligned}
|J_2|\frac{\Delta}{\Delta+1} &> |J_2|\frac{\Delta+1}{\Delta}\frac{m}{m+1} \\
&= \frac{\Delta+1}{\Delta}\frac{|J_2|}{1+1/m} \\
&> \frac{\Delta+1}{\Delta}\frac{|J_2|}{1+1/Q} \\
&= \frac{\Delta+1}{\Delta}Q\frac{|J_2|}{Q+1} \\
&\geq \frac{\Delta+1}{\Delta}Q
\end{aligned}.$$

Contradiction!

Therefore, for any solution of value at least $L \cdot |T_2| + \frac{1}{1+Q}$, we can find a set cover J of size Q.

□

Combining the two directions, the reduction is completed.

Chapter 6

Proof of Theorem 3.4

Recall the All-Pairs Inverse Shortest Path Problem:

Given: a graph $G = (V, E)$, where V is the vertex set and E is the edge set, and a set of paths of G: $\{P_{ij} \subset E : \forall i, j \in V, i \neq j\}$, a positive number D.

Find: edge weights $\lambda_e \in \mathbb{N}_+, e \in E$, s.t. $\lambda_e \leq D$, and P_{ij} is the shortest path between i and j. (The shortest paths are not necessarily unique.).

Theorem 3.4 claims that the All-Pairs Inverse Shortest Path Problem is NP-Hard.

We will use a reduction from the set partitioning problem where each set has exactly 3 elements and each element is contained in exactly 3 sets defined as follows (RX3C):

Given: A set of n elements, where n is a multiple of 3, $S := \{1, 2, \ldots, n\}$, and n subsets $S_j, j = 1, 2, \ldots, n$ of S such that each element is contained in exactly 3 subsets and each subset consists of exactly 3 elements of S, i.e., $|\{j : i \in S_j\}| = 3, \forall i = 1, 2, \ldots, n$ and $|S_j| = 3, \forall j$.

Determine: if there exists an exact partition of S, i.e., $\exists I \subseteq \{1, 2, \ldots, n\}$ s.t.

$$\begin{aligned} &\bigcup_{i \in I} S_i = S \\ &S_i \cap S_j = \emptyset, \forall i, j \end{aligned} .$$

RX3C is NP-Hard [29]. Our proof is inspired by Bley's proof [14]

In order to complete the reduction, we will construct an instance of All-Pairs Inverse Shortest Path instance for a given RX3C instance, and show that we can choose the value of D so that we can use the result of the All-Pairs Inverse Shortest Path instance to determine whether the given set in RX3C is partitionable.

6.1 Construction of the instance

In the following section, we will construct the graph instance of All-Pair Inverse Shortest Path for the corresponding RX3C instance step by step, and in each step we will introduce one graph component and identify the shortest path between each pair of nodes in V. We will fix certain arc weights, then discuss the possible weights of all the remaining arcs in order to route the prescribed shortest paths and how they will affect the objective in different situations.

Before we introduce our graph components, we first define the following operation:

Definition 6.1 (Attach a Chain)**.** Let $u, v \in E$ be an arc, we can "attach a chain" of arcs of length $26n - 1$ to v, namely the root of the chain. The other end of the chain, denoted by w, will be connected to u and the whole graph component $u \to v \to w \to u$ forms a cycle, as shown in Figure 6.1. We will set the arc (u, v) plus the chain $v \to w$ to be the shortest path between u and w.

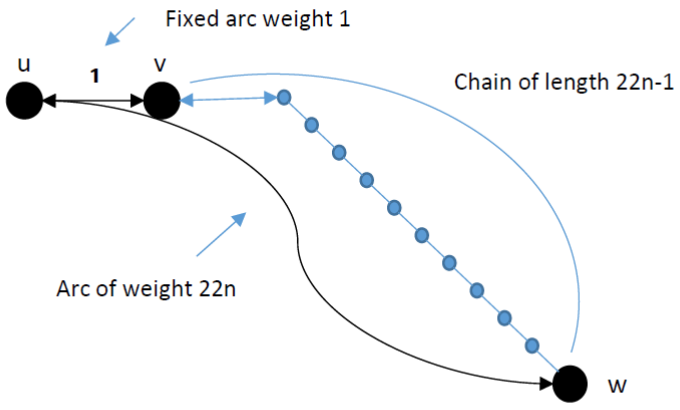

FIGURE 6.1: Attach a Chain

With the definition above, we have the following proposition:

Proposition 6.2. *Suppose that there exists a feasible solution λ to an All-Pairs Inverse Shortest Path instance where* $\max\{\lambda_e : e \in E\} = 26n$*, and uv is an arc applied with an "attach-a-chain" procedure, then $\lambda_{uv} = 1$.*

Proof. Since the maximum arc weight is $26n$, the arc uw must have weight $\leq 26n$. Since the path $u \to v \to w$ is the shortest paths between u and w, we know that this path must have weight $\leq 26n$. Since there are $26n$ arcs in this path, we know that the arc uv must have weight 1, since $\lambda_e \in \mathbb{N}_+, \forall e \in E$. In fact, all the arcs on the chain $v \to w$ have weight 1. □

Note that this "attach-a-chain" procedure can be used in a similar fashion onto a chain of arcs to ensure all the arcs have weight 1, which can be seen from the following proposition.

Proposition 6.3 (Corollary of Proposition 6.2)**.** *Suppose we have a chain of arcs $u \to u_1 \to u_2 \to \ldots \to u_{s-1} \to v = u_s$, we can attach a chain $u_{s+1} \to \ldots w$ to u_s (root) of length $26n - s$ and set $u \to u_1 \to \ldots w$ to be the shortest path, and finally link w back to u. Then for any feasible solution λ where the maximum arc weight is equal to $26n$, each arc in the chain $u \to u_1 \to u_2 \to \ldots \to u_s = v$ has weight 1.*

This revised version of "attach-a-chain" is going to be used to ensure the arcs on the widget-linking arc sequences, to be introduced later, with weight 1. We will call this type of chain added through "attach-a-chain" procedure the "attached chain".

6.1.1 Graph Components

From Windows to single Widget. First, for each element $i \in S$, we construct a graph component called "The Widget" with 13 nodes.

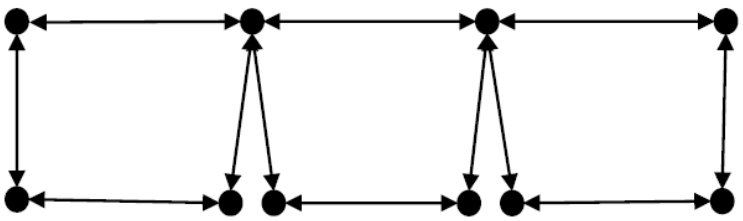

FIGURE 6.2: Windows

Figure 6.2 illustrates "The Windows", the basic unit structure of "The Widgets". There are 6 vertical arcs, and 6 horizontal arcs. We name the windows the Left Window (LW), the Middle Window (MW) and the Right Window (RW) from left to right respectively. We call the 3 arcs on the top the "level 2" arcs, and ones in the bottom the "level 1" arcs. The nodes on "level 1" and "level 2" arcs are called "level 1" and "level 2" nodes respectively. From Figure 6.2 we can see that there are 4 level 2 nodes and 6 level 1 nodes. In principle, we map each piece of "The Windows" to a component in the given RX3C instance. Each widget created above represents an element k in S. It contains 3 windows and each window represents the set that contains k. The order of the sets does not matter. Note that there are exactly 3 sets that contains each element. Figure 6.3 Shows the definitions within a window. The four nodes within a window are called the top left node, the top right node, the bottom left node and the bottom right node according to its physical position in the window, respectively.

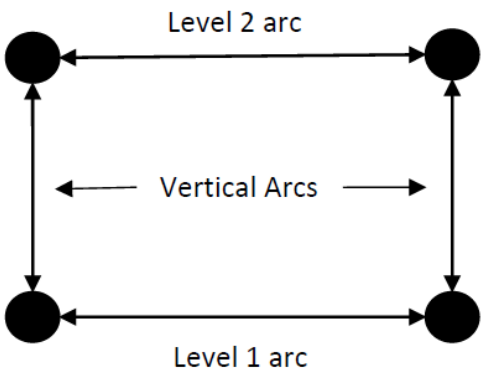

FIGURE 6.3: Window

Note that within a widget, there are two level 2 nodes (the 2 non-corner nodes out of the four) that are shared between windows. In addition we denote TLC to be the top left corner of LW and TRC to be the top right corner of RW for each widget.

We apply the "attach a chain" procedure along all the vertical arcs, adding more nodes and arcs to "The Windows".

In addition to the three windows, we will add a chain of 4 arcs (including 3 additional nodes) from TLC to TRC of a widget and call it "the roof" shown in Figure 6.4. This forms the entire 13 nodes Widget (plus the "attached chain").

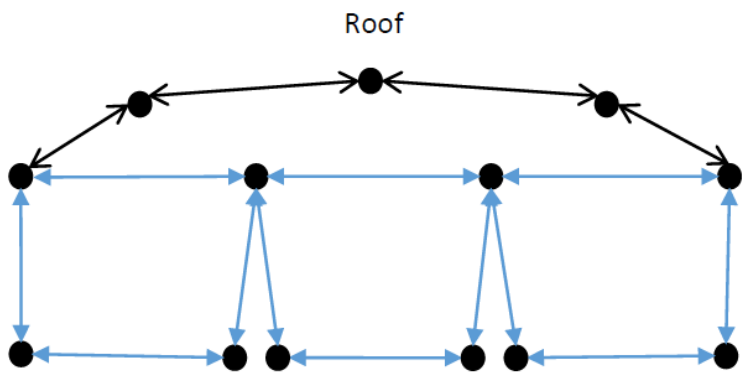

FIGURE 6.4: Widget

The Widgets Finally, we create n (the parameter of the given RX3C instance) such widgets, naming them $W_1, W_2, \ldots W_n$, as the number of widgets equals to the number of elements (sets) in the RX3C instance. At the same time, we connect W_i to W_{i+1} for $i \in \{1, \ldots, n-1\}$ by contracting the TRC of W_i and TLC of W_{i+1} for $i = 1, 2, \ldots n-1$ to form a single connected component, called "The Widgets", and we add to the TRC of W_n a chain of arcs of length $26n - 4n = 22n$, called it the "tail chain." We apply the revised "attach-a-chain" procedure, as stated in Lemma 6.3, to the tail chain to ensure that each arc on the tail chain has weight 1 given that the max arc weight is $26n$. At last, we add an arc from the TLC of the first widget to the end of the tail chain . In Figure 6.5, the dotted lines between each TRC and TLC are used to indicate that the two nodes are actually contracted, and the tail chain is represented by the dot-dash line.

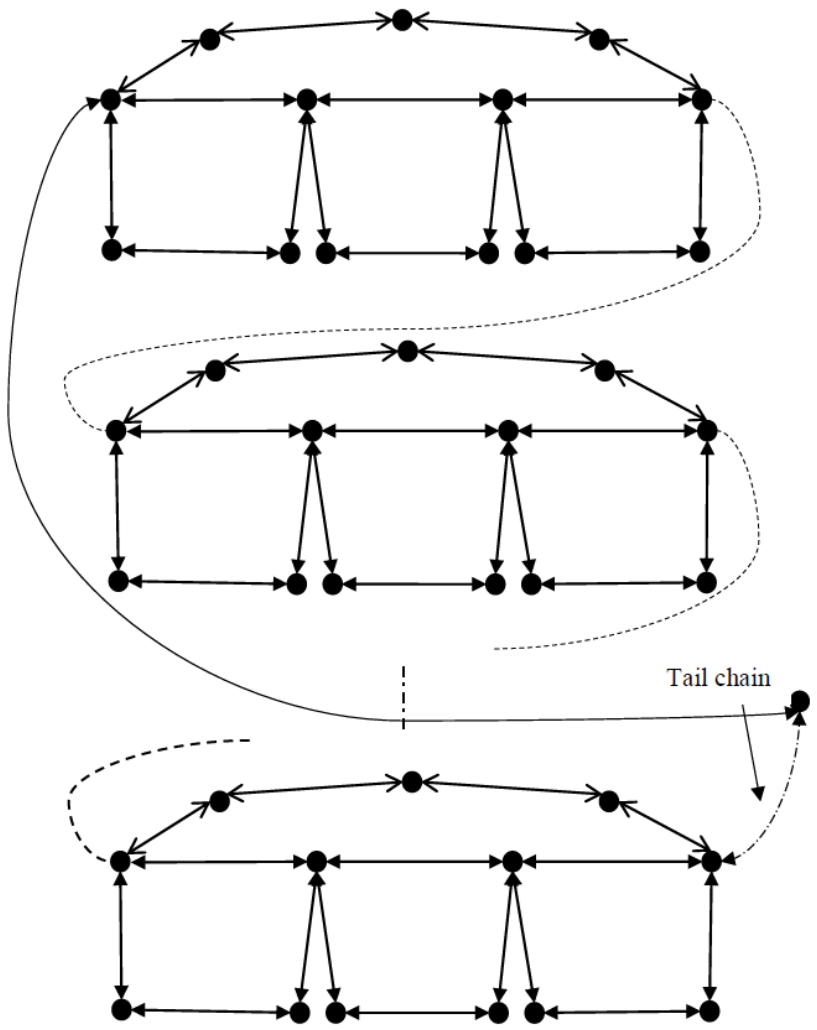

FIGURE 6.5: The Widgets

Definition 6.4 (Ahead Of)**.** We say Widget W_i is "ahead of" Widget W_j if $i < j$.

6.1.2 Prescribed Shortest Paths in "The Widgets"

In all the figures in this section, the thick arcs represent the prescribed shortest path and the thick dotted arcs represent a close alternative.

Since all of the arc weights have to be positive integers, so for any feasible solution $\lambda \in \mathbb{N}^{|A|}$ to the All-Pair Inverse Shortest Path Problem, we have $\lambda_e \geq 1, \forall e \in E$, and we define $\lambda_P = \sum_{e \in P} \lambda_e$ for the path P.

First, let $\lambda \in \mathbb{N}^{|A|}$ be a feasible solution to the instance. We consider the TLC and the TRC of a single widget. There are two paths between the two nodes within the

widget. One is the roof, with 4 arcs, and the other is path of all level 2 arcs, namely the "subroof" path, with 3 arcs. We set the roof to be the shortest, and we have the following propositions.

Proposition 6.5. *Suppose λ is feasible to the All-Pairs Inverse Shortest Path instance, then $\lambda_{\text{subroof}} \geq \lambda_{\text{roof}} \geq 4$.*

Proof. Since roof is the shortest path between TLC and TRC for each widget, we know that $\lambda_{\text{subroof}} \geq \lambda_{\text{roof}}$. Moreover, since the arc weight is limited to positive integer only, so the path weight is at least the same as the hop count, and note that there are four arc in a roof, we know that $\lambda_{\text{roof}} \geq 4$. □

Proposition 6.6. *Suppose that λ is feasible to All-Pairs Inverse Shortest Path Problem, and Suppose that $\lambda_{\text{subroof}} \leq 4$, then one of the level 2 arc must have weight 2 and the other two level 2 arcs have weight 1.*

Proof. By Proposition 6.5, we know that $\lambda_{\text{subroof}} \geq \lambda_{\text{roof}} \geq 4$. Since $\lambda_{\text{subroof}} \leq 4$, we know that

$$\lambda_{\text{roof}} = \lambda_{\text{subroof}} = 4.$$

On a subroof, there are 3 arcs, and since each arc has weight at least 1, we must have two arcs of the 3 with weight 1 and the other one with weight 2. □

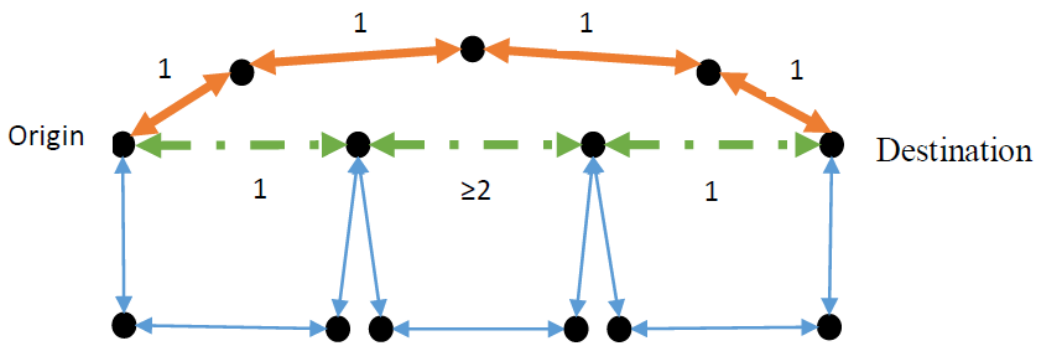

FIGURE 6.6: Roof Constraints

Second, for each window, there are two paths from the top left corner to bottom right corner. We set the one that uses the level 2 arc to be the shortest (indicated in

Figure 6.7 and Figure 6.8 below). Since all the vertical arcs have weight 1 (enforced by "attach-a-chain" procedure), we know that within a window, the level 1 arc has at least the same weight as the level 2 arc, i.e. $\lambda_{l1} \geq \lambda_{l2}$. In the first case below, the level 1 arc is forced to be at least 1 while in the second case below, the level 1 arc is forced to be at least 2. Therefore, we have the following proposition.

Proposition 6.7. *Suppose that the objective is* $26n$*, let* $\lambda \in \mathbb{N}^{|A|}$ *be a feasible solution to the instance, then for each window,* $\lambda_{l1} \geq \lambda_{l2}$*.*

Proof. By Proposition 6.2, we know that the vertical arcs have weight 1. Consider the two paths from top left corner to the bottom right corner:

- origin $\xrightarrow{\text{lvl 2 arc}}$ top right corner $\xrightarrow{\text{vertical arc}}$ destination
- origin $\xrightarrow{\text{vertical arc}}$ bottom left corner $\xrightarrow{\text{lvl 2 arc}}$ destination

Since the vertical arcs have the same weight, and from the assumptions we know that the path that uses the level 2 arc is the shortest. Thus, we can conclude that the level 2 arc has weight at most the same weight as the level 1 arc. i.e.

$$\lambda_{l1} \geq \lambda_{l2}.$$

□

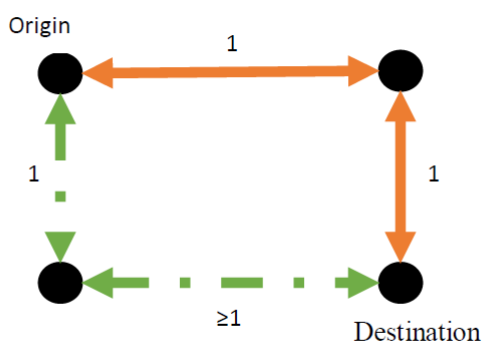

FIGURE 6.7: Window Constraints: Case 1

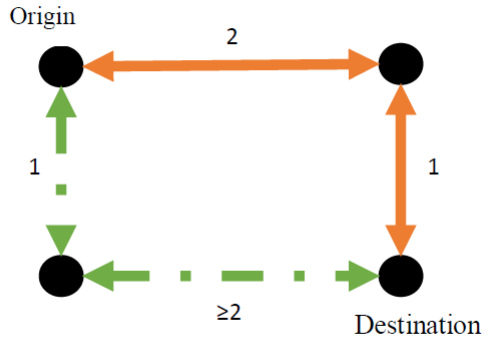

FIGURE 6.8: Window Constraints: Case 2

Finally, consider the TLC of W_1 and end of the tail chain. We set the path that goes through all and only the level 2 arcs plus the tail chain to be the shortest between these two nodes, and we call the piece without the tail chain "The Path". By Proposition 6.5 we know that the weight of the long level 2 path is precisely the union of all level 2 subroof from all n widgets, therefore we know $\lambda_{\text{the}_\text{p}\text{ath}} \geq 4n$, where $\lambda_{\text{the}_\text{p}\text{ath}}$ is the weight of "The Path", and thus, "The Path" plus the tail chain (with weight $22n$) has value at least $26n$. Since the alternative path connecting the TLC of W_1 and the end of the tail chain is the single arc that joins the two nodes, namely "The Arc", with weight $\lambda_{\text{the}_\text{a}\text{rc}} \geq 26n$, where $\lambda_{\text{the}_\text{a}\text{rc}}$ is the weight of "The Arc". Thus the objective value is at least $26n$.

Proposition 6.8. *The objective value of the derived All-Pair Inverse Shortest Path instance from the given RX3C instance, is at least* $26n$.

Proof. By Proposition 6.5 we know that each of the subroof has weight at least 4. Since "The Path" consists of n subroofs, with weight at least $4n$. There are an additional tail chain of $22n$ arcs, and note that there is an arc connecting TLC of W_1 (one end of "The Path") to the end of the tail chain, and this arc should have no less weight than "The Path" plus the tail chain, therefore, it must have at least $4n + 22n = 26n$. Thus the objective value, i.e., the maximum arc weight, is at least $26n$. □

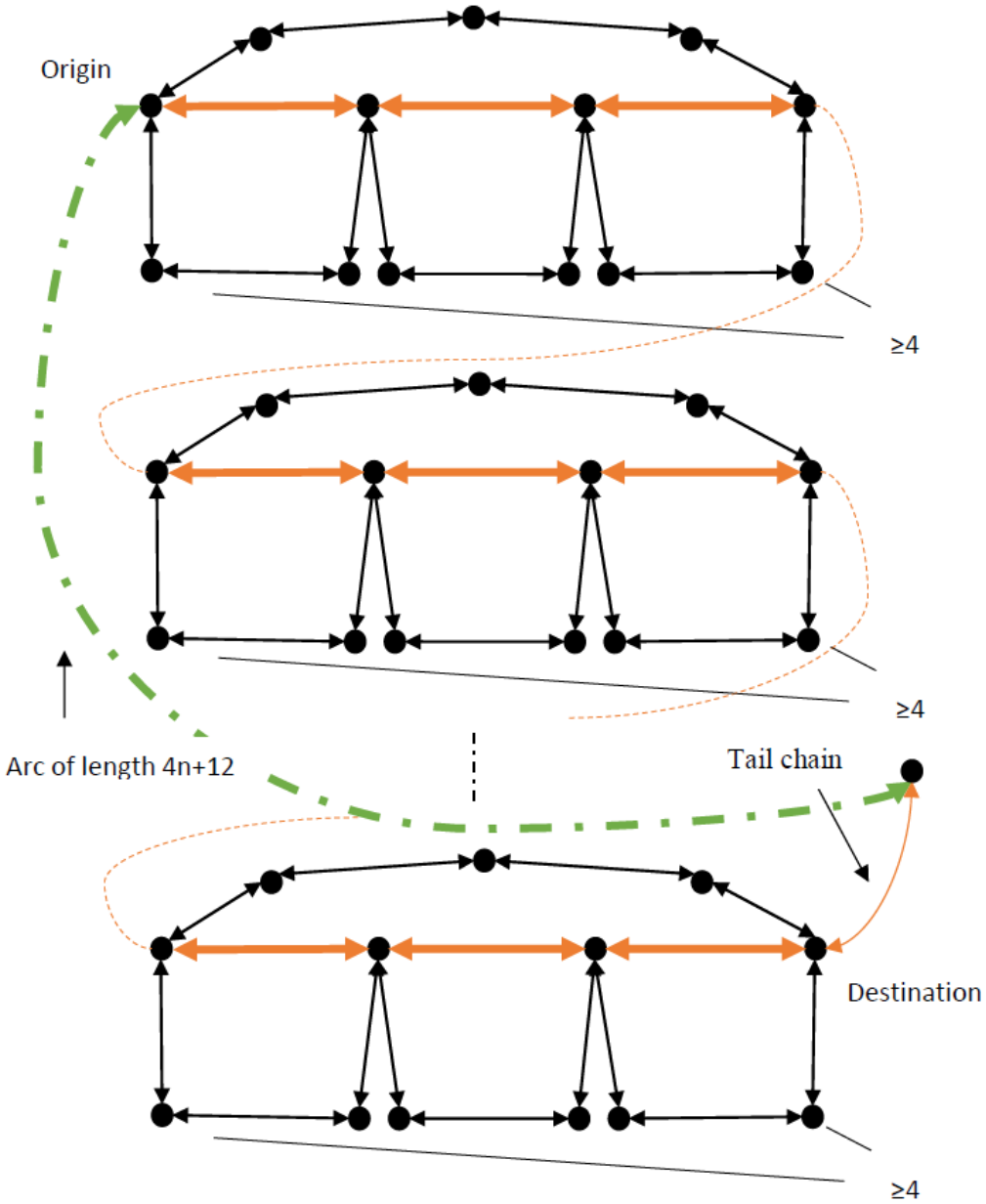

FIGURE 6.9: Longest Arc of Weight 4n

So far, we have introduced The Widgets as the main graph component of the problem instance. With The Widgets, we translate the master set into a collection of elements, each of which are represented as a Widget. Within each Widget, we use three windows to represent the subsets that contain this element. However, windows from different widget that represents the same set has not yet been depicted at this point. This will be resolved in the next section.

6.1.3 Widget-Linking Arc Sequences

There is one more set of chains to add to The Widgets: the "widget-linking arc sequences". First we connect The Widgets to the RX3C instance by letting each widget in The Widgets correspond to an element in the given RX3C instance and each window correspond to a subset that contains the element. Thus, each widget has three windows, which corresponds to the fact that each element is contained in exactly 3 sets. However, we haven't had a way to incorporate the fact that each set contains exactly three elements. In the following section, we will describe how to connect the windows from different widgets that corresponds to the same set. We do that in the following way. We first define two terms, Relative Position and Position Difference, as follows:

Definition 6.9 (Relative Position)**.** The relative position of a window in a widget W, denoted by relpos(W), is the location of the window in a widget:

- 1 for the left position (LW);
- 2 for the center position (MW);
- 3 for the right position (RW).

Definition 6.10 (Position Difference)**.** In The Widgets, the Position Difference between two windows W_i and W_j, $i < j$, denoted by posdef(W_i, W_j), is a value that measures the length of the path between the corresponding nodes in two windows. The actual value depends on the relative position of the two windows.

Let $m = j - i$, then if

- W_i and W_j are in the same relative position: posdif$(W_i, W_j) = 4m$;
- relpos(W_i) − relpos$(W_j) = 1$: posdif$(W_i, W_j) = 4m - 2$;
- relpos(W_i) − relpos$(W_j) = 2$: posdif$(W_i, W_j) = 4m - 3$;
- relpos(W_i) − relpos$(W_j) = -1$: posdif$(W_i, W_j) = 4m + 1$;
- relpos(W_i) − relpos$(W_j) = -2$: posdif$(W_i, W_j) = 4m + 2$.

Since each set contains 3 elements, windows that corresponds to the same set appear in 3 different widgets. From W_1 to W_n, the 3 windows occur in an order, say in widgets W_i, W_j, W_k, where $i < j < k$, and we call W_i the leading window. We will add four arc sequences between the two windows in W_i and W_j, and between the two windows in W_i and W_k in the following manner: the top left corner connects to the bottom left corner, the top right corner connects to the bottom right corner, and so on. The number of arcs along the arc sequences equals to the position difference of the two windows. The following Figures illustrates all the cases.

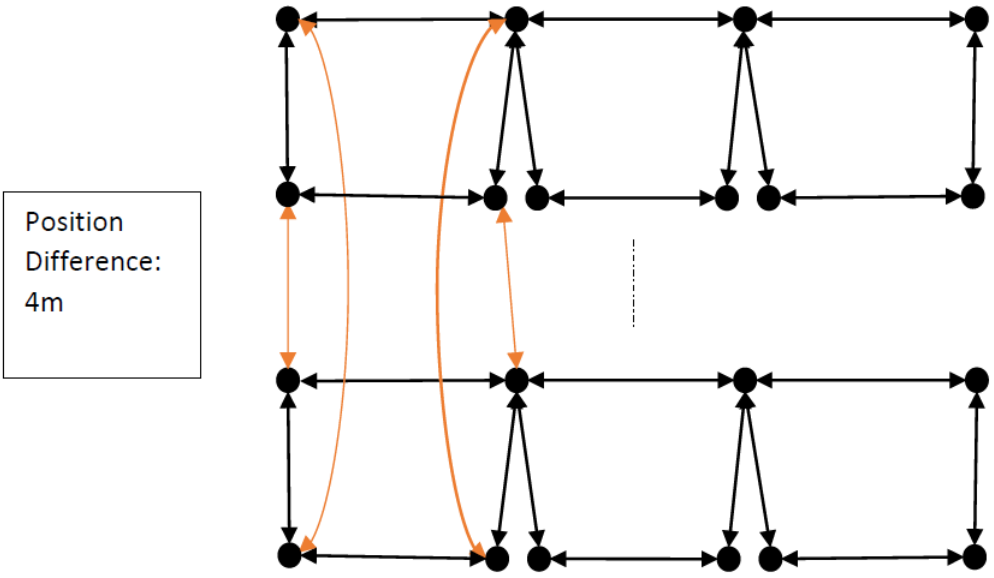

FIGURE 6.10: Case 1

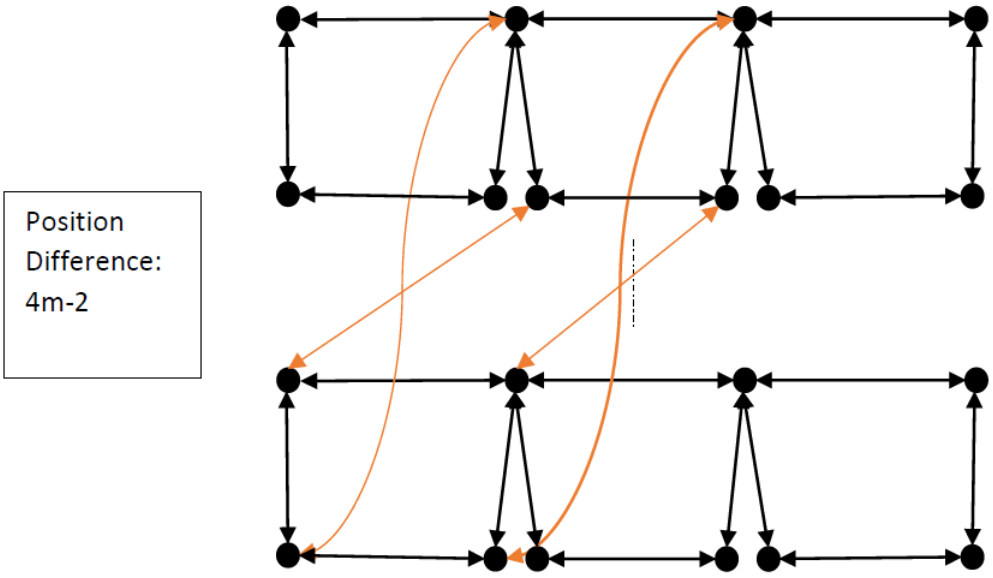

FIGURE 6.11: Case 2

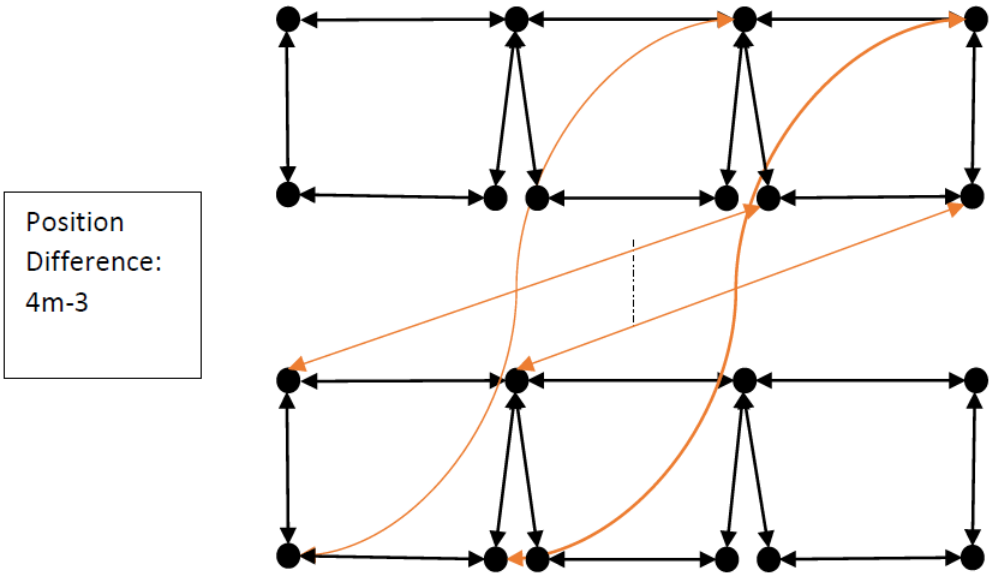

FIGURE 6.12: Case 3

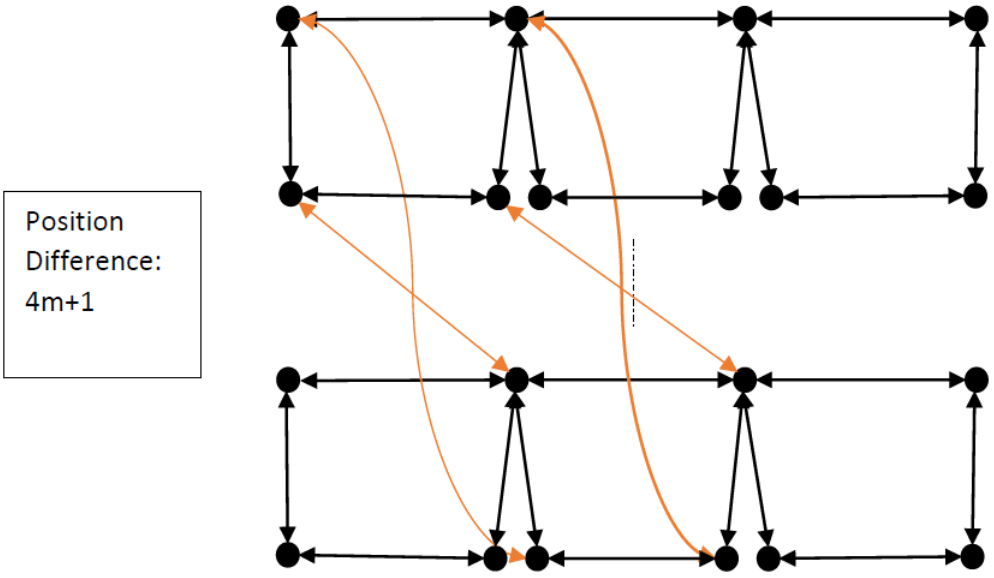

FIGURE 6.13: Case 4

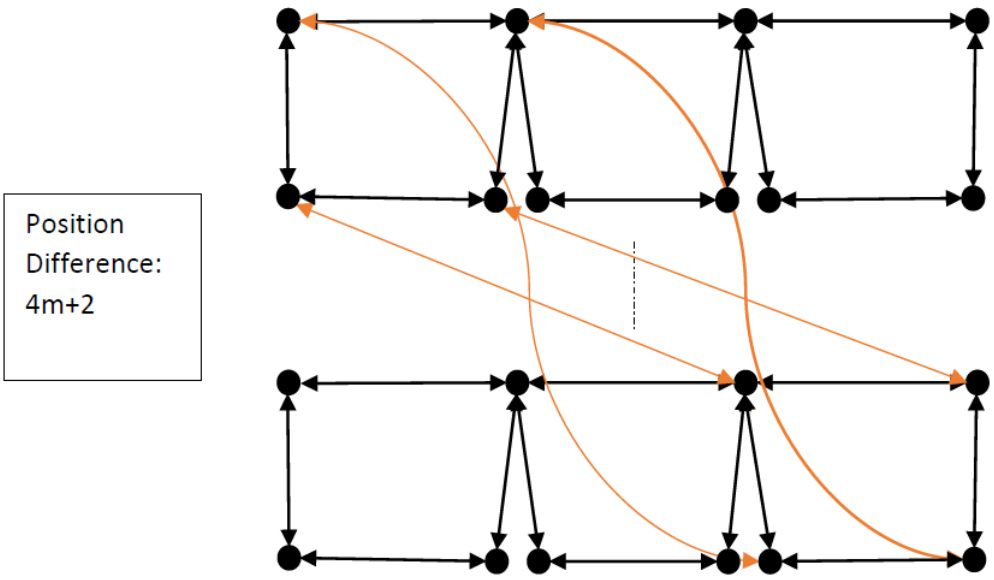

FIGURE 6.14: Case 5

With the definition above, notice that for each pair of linked windows, the two pairs of widget-linking arc sequences have the same number of arcs. Consider two windows that are linked by four widget-linking arc sequences, and the paths between the top

left corner of one window and bottom right corner of the other window. Among all paths that connect the two end points, we will focus on two possible shortest paths: one is to go through the chain first and then use the level 1 arc; the other is to go through the level 2 arc first and then go through the chain. We set the path that use the level 1 arc be the shortest. Note that from the definition of the widget-linking arc sequence, the two chains have the same number of arcs, and they are enforced with weight 1 by the "attach-a-chain" procedure, and thus the two chains have the same weight, which leads to the following proposition.

Proposition 6.11. *Let $\lambda \in \mathbb{N}^R$ be a feasible solution to the instance. For two windows that are linked by widget-linking arc sequence, let a, b, c, d represent the level 1 arc of the first window, the level 2 arc of the first window, the level 1 arc of the second window and the level 2 arc of the second window, respectively, then $\lambda_a \leq \lambda_d$ and $\lambda_c \leq \lambda_b$.*

Proof. Since $W1$ and $W2$ are linked, we know that the shortest path between the top left corner of $W1$ and the bottom right corner of $W2$ is the one that goes through the widget-linking arc sequence and then goes through the level 1 arc. Alternative to this path, we can also go through the level 2 arc first and then reach the destination through the other widget-linking arc sequence (Note that widget-linking arc sequence always exist in a pair). Since any widget-linking chain pairs between $W1$ and $W2$ have the same weight by definition, we know that the level 1 arc of $W1$ is less than or equal to the level 2 arc of $W2$. Similarly, we know that the level 1 arc of $W2$ is less than or equal to the level 2 arc of $W1$, as desired. □

Combine Proposition 6.7 and Proposition 6.11 we have the following:

Proposition 6.12 (Corollary of Proposition 6.7 and Proposition 6.11)**.** *Let $\lambda \in \mathbb{N}^{|A|}$ be a feasible solution to the instance. Suppose that the objective is $26n$. For each window, the level 1 arc and the level 2 arc has the same weight. Furthermore, for windows $W1$ and $W2$ that are linked by widget-linking arc sequence, let a, b, c, d represent the level 1 arc of the first window, the level 2 arc of the first window, the level 1 arc of the second window and the level 2 arc of the second window, respectively, $\lambda_{\mathrm{a}} = \lambda_{\mathrm{b}} = \lambda_{\mathrm{c}} = \lambda_{\mathrm{d}}$.*

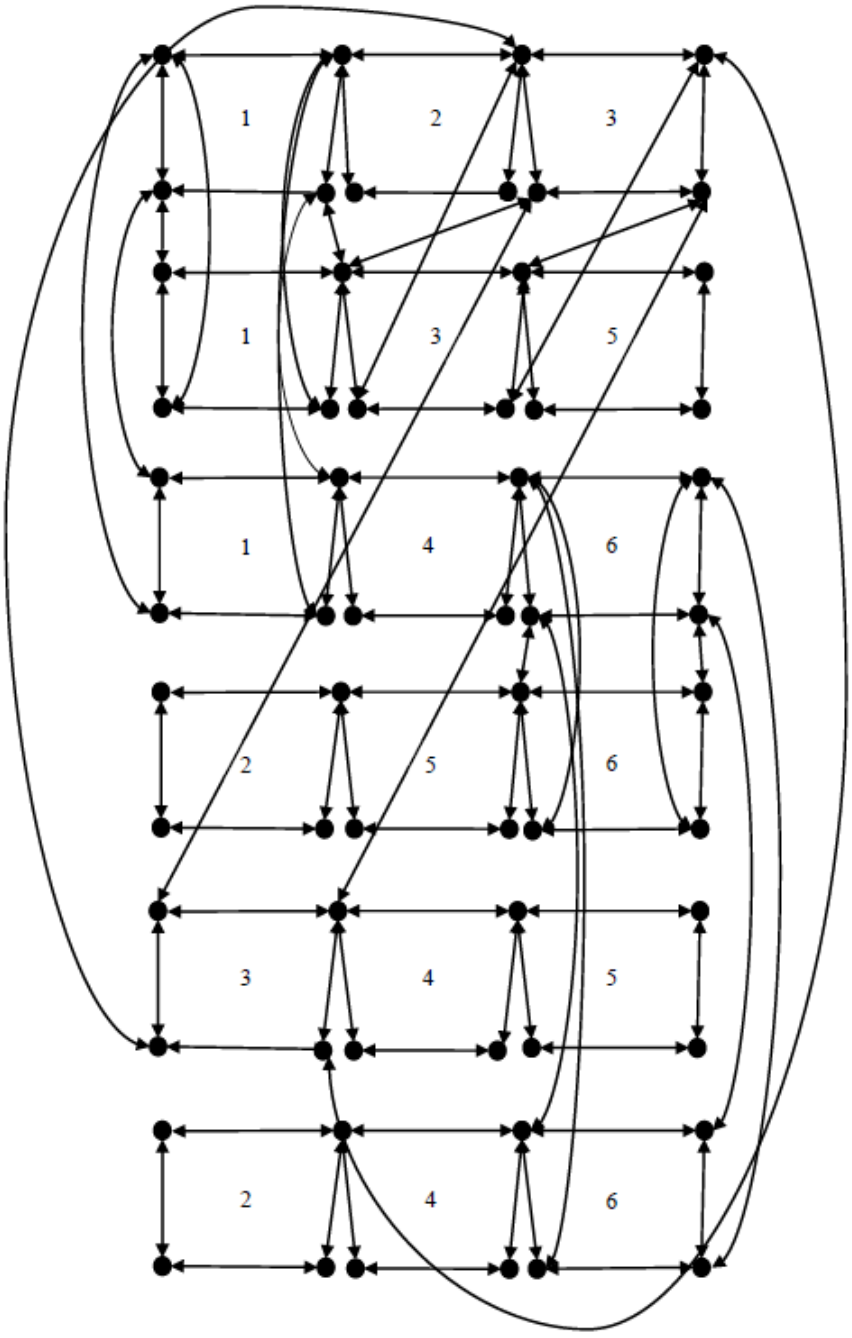

FIGURE 6.15: An Example

Figure 6.15 shows the graph corresponding to an RX3C instance of 6 elements. The set $S = \{1, 2, 3, 4, 5, 6\}$, the subsets are

- $S_1 = \{1, 2, 3\}$,
- $S_2 = \{1, 4, 6\}$,

- $S_3 = \{1, 2, 5\}$,
- $S_4 = \{3, 5, 6\}$,
- $S_5 = \{2, 4, 5\}$,
- $S_6 = \{3, 4, 6\}$.

The widget-linking arc sequences for element 2,4,5 are omitted. Note that the number in the center of each window indicate the set it represents.

6.1.4 Prescribed Shortest Path

So far, we have defined the prescribed shortest paths for the following origin destination pair:

- TLC and TRC of each widget (the roof);
- TLC of W_1 and the end of the tail chain (The Path plus the tail chain);
- Top left corner and bottom right corner of each window (the two-hop paths that containst the level 2 arc);
- Top left corner of W_i and bottom right corner of W_j where W_i and W_j are linked by widget-linking arc sequence (the "two-hop" paths that contains the level 1 arc).

For any pair of nodes (A, B) where at least one end, say A, lies on the attached chain or the tail chain, the shortest path is defined to be the partial chain from A to the root of the chain C, plus the shortest path between B and C, given that the all-pairs shortest paths are defined. Hence, there is only one more case left, for which we will do the following: for all the remaining pair of nodes, we add an arc between the nodes and set the single arc to be the prescribed shortest path between its two end nodes.

With the addition of these arcs, and let $D = 26n$, we have finished the construction of the instance, including a graph (network), the prescribed shortest path between all pairs of nodes and the constant D.

6.2 Proof: Part I

We will show in this section that if the objective value of the constructed instance is $26n$, the corresponding RX3C instance is partitionable

By Proposition 6.8, we know that the objective value is at least $26n$. Suppose that the objective value is $26n$. By Proposition 6.2 and Lemma 6.3, we know that all the vertical arcs in the windows have weight 1, and all the arcs on widget-linking arc sequences have weight 1, because of the "attach-a-chain" procedure. It also implies that "The Path" plus the tail chain has weight equal to $26n$ and thus "The Path" has weight $4n$, so that all the subroofs of each widget have exactly weight 4. By Proposition 6.6, we know that each widget will have exactly one level 2 arc with weight 2, and the other ones with weight 1.

To show that the corresponding RX3C instance is partitionable, we select the sets that correspond to the windows with both level 1 and level 2 arcs assigned with weight 2. By Proposition 6.12, we know that all the windows that are linked have the same weight assigned to all of their level 1 and level 2 arcs . Therefore, this selection is a partition because

1. Each widget has exactly one window selected, which means that every element is covered;
2. The intersection of any two selected sets will be empty, since otherwise the windows corresponding to these two sets will be in a same widget that represents the element they share, and thus this widget has two windows whose horizontal arcs are assigned with weight 2, contradiction.

Therefore we proved that if the objective value is $26n$, the RX3C instance is partitionable.

6.3 Proof: Part II

Suppose a given RX3C instance is partitionable. We will show that we can assign weights to the arcs so that the All-Pair Inverse Shortest Path Problem has solutions with objective value $26n$.

Proof Sketch: We know from the construction of the graph component that for any feasible solution λ, "The Path" plus the tail chain has value at least $26n$ by Proposition 6.8, and therefore since the path is the shortest, we can conclude that any feasible solution "The Arc" has weight at least $26n$. We will prove that the solution we construct below is feasible, and any other arc in the graph has weight no larger than $26n$, which concludes that the objective value is $26n$, and thus we are done.

Let I be a partition, the level 1 and level 2 arcs of the windows represent the sets in I have weight 2. Since I is a partition, each widget has 1 and only has 1 window assigned with weight 2. We then let "The Arc" have weight $26n$ and every other arc that are on "The Widget", the attached chains, the tail chain and the widget-linking arc sequences, have weight 1. Let $\overline{G}$ be the subgraph of G with all the weighted arcs. For each of the remaining arcs, the weight is equal to the weight of the shortest path between the corresponding end points in $\overline{G}$. Clearly this weight-assigning procedure is done in polynomial time.

Now there are two steps left to complete the proof: first, verifying the shortest paths (feasibility); second, showing that the objective value is indeed $26n$.

6.3.1 Verifying the Prescribed Shortest Paths and All the Other Paths

6.3.1.1 The Path

Between the TLC of W_1 and the TRC of W_n, we have "The Arc" as well as "The Path" plus the tail chain. We want "The Path" plus the tail chain to be the shortest.

Consider any path between TLC (of W_1) and TRC (of W_n). We can partition the path into a union of paths whose start and end nodes are both at level 2. We want this partition to have as many subpaths as possible. Then there are only two types of paths: (note that the arcs on "attached chain" will never be considered since using them we are adding an extra cost of $22n$, which is undesirable).

- Type I Path: Paths with only level 2 arcs;
- Type II Path: Paths that go to level 1 arcs and stay in level 1 until finally going back to level 2.

Suppose we can prove that the Type II path can always be replaced by Type I path with the same start and end nodes, then it means the shortest path must consist of only the level 2 nodes, and therefore, the shortest path is "The Path" plus the tail chain.

Consider a Type II Path. It could either be within one widget or with at least two widgets involved. If it is within one widget, it must go down to level 1, and then go through in the level 1 arc and then go back up to level 2. Since level 1 arcs have the same weight as level 2, this path will have weight 2 more than the corresponding Type I path, since it travels through the vertical arcs twice, once up and once down, inccurs 2 extra units of cost.

Consider the path that has two widgets involved. Without loss of generality, we assume that the two widgets are adjacent to each other, otherwise, the length of both Type I and Type II paths for the same start and end nodes will increase by a same constant (4 times the number of widgets in between). There are 16 cases, which are shown in the following figures, and these are an enumeration of all cases because it is essentially a Cartesian product of all the level 2 nodes of one widget and all the level 1 nodes (equivalent nodes which connects to the same level 2 node by a vertical arc count as 1 nodes), which consists of $4 \times 4 = 16$ cases. In the following figures, bold path represents the type 2 path and the bold dotted path represents the level 2 path; A is the start node and B is the end node. Without loss of generality, we only include the case where the path starts from node A and goes through the arc sequence to the other widget, and goes up to node B. Alternatively we may choose to start the path AB from going down to level 1, and use the arc sequence to reach B, which has exactly the same cost as the one we choose above, due to the fact that the arc sequence that connects the same pair of windows has the same length. Therefore, we can omit the discussion of it. Also note that for Case 2, 3, 6, 7, 10, 11, 14, 15 below, there are two possible routes, and the arc sequences these routes use have the same length, because the position difference is the same. Most cases that involve either TLC or TRC may not seem to exist, but considering the assumption that TRC and TLC from consecutive widgets are the same nodes, which realize all the cases, and it is explained in Figure 6.32.

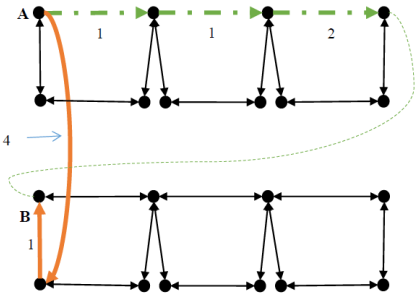

FIGURE 6.16: Case 1

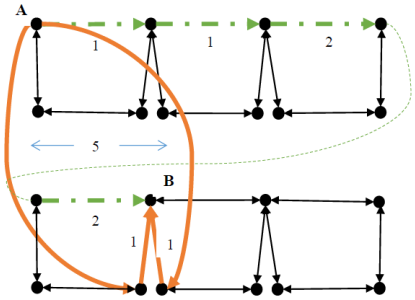

FIGURE 6.17: Case 2

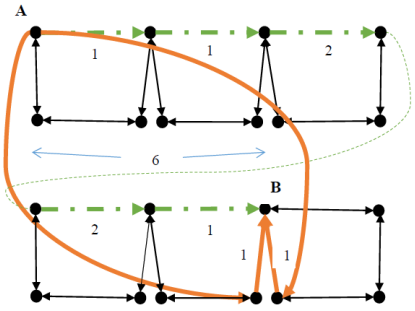

FIGURE 6.18: Case 3

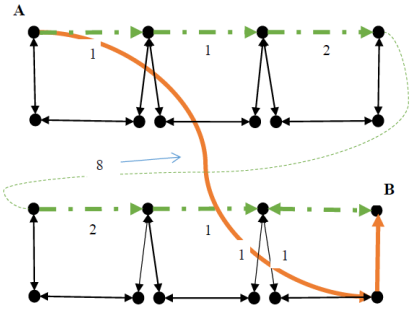

FIGURE 6.19: Case 4

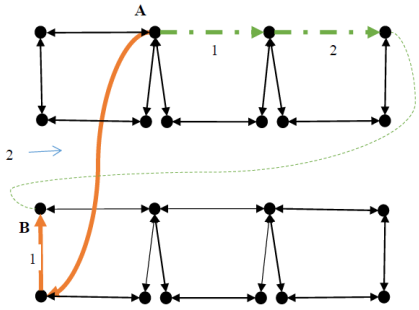

FIGURE 6.20: Case 5

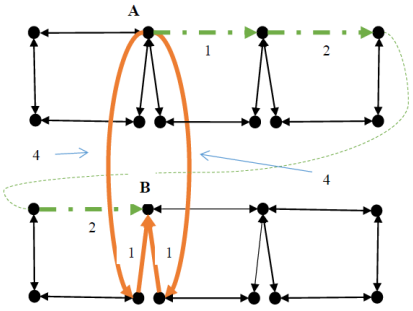

FIGURE 6.21: Case 6

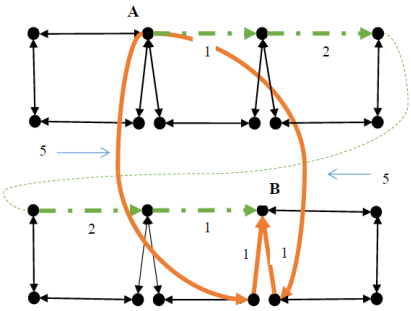

FIGURE 6.22: Case 7

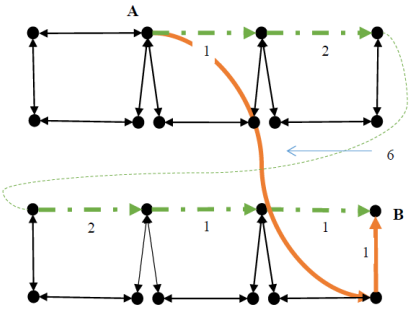

FIGURE 6.23: Case 8

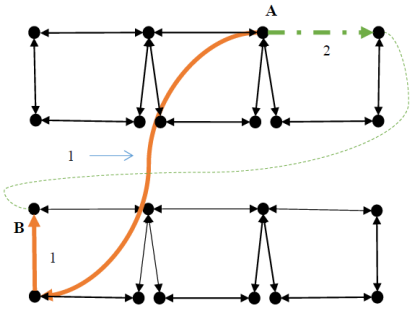

FIGURE 6.24: Case 9

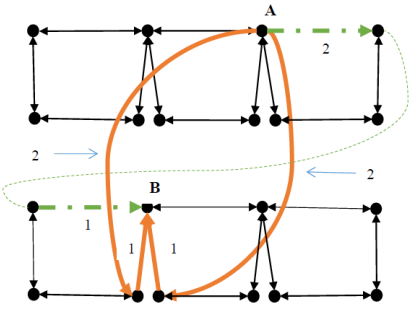

FIGURE 6.25: Case 10

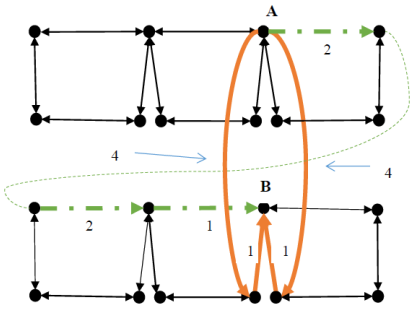

FIGURE 6.26: Case 11

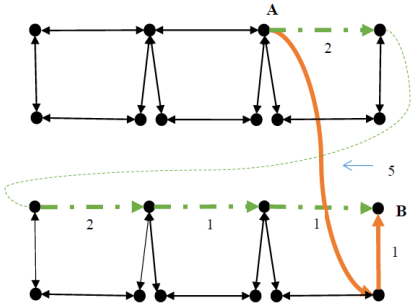

FIGURE 6.27: Case 12

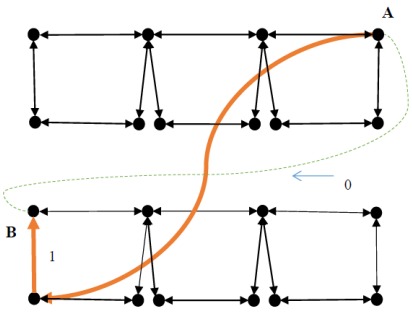

FIGURE 6.28: Case 13

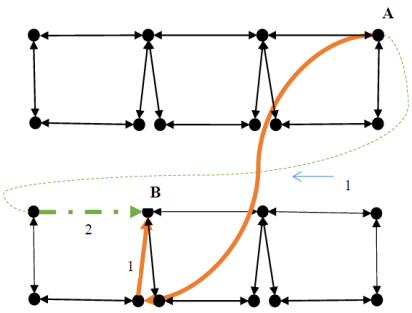

FIGURE 6.29: Case 14

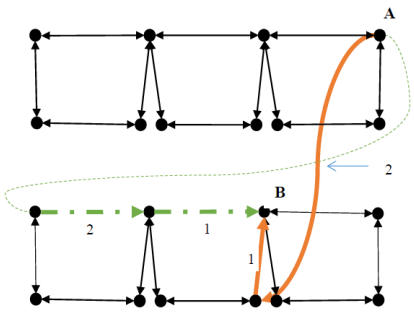

FIGURE 6.30: Case 15

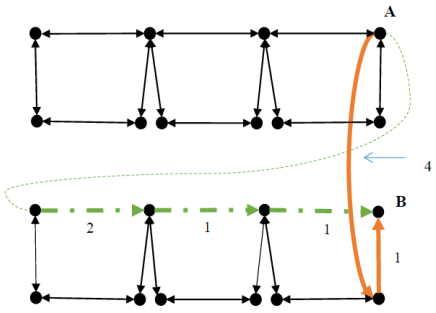

FIGURE 6.31: Case 16

Note that the last four cases are actually equivalent to the first four cases since the top right corner is essentially the top left corner of the next widget as shown in the Figure 6.32:

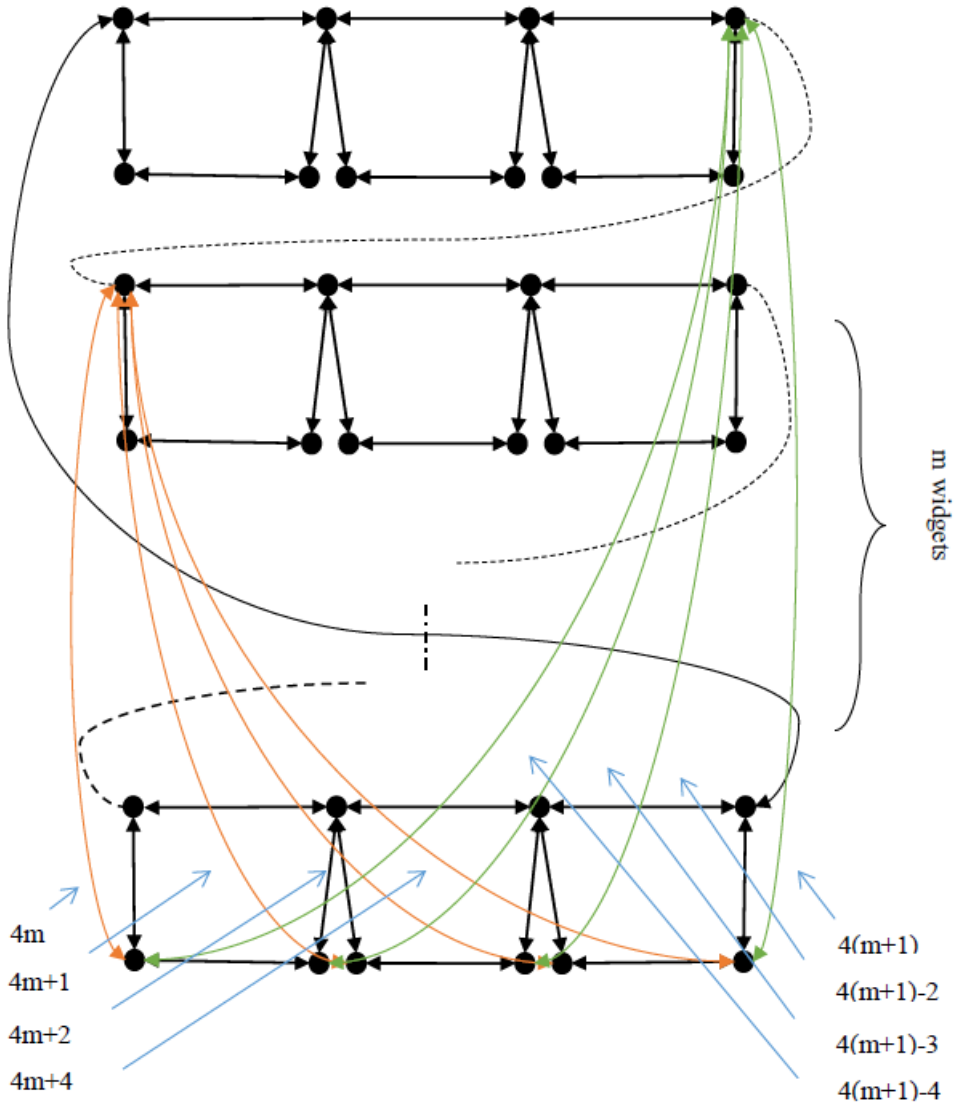

FIGURE 6.32: Equivalence of the Case 1-4 with Case 13-16

The graph above also proves that the definition of the position difference, reflected by the length of the arc sequences, is consistent with the fact that the TRC of W_i is the same as TLC of W_{i+1} for all $i = 1, 2, \ldots, n-1$.

For all of the 16 cases shown above, we can see that each of the dotted path has weight less than or equal to the thick path by the construction of the widget-linking arc sequences. Therefore, we can conclude that "The Path" is the shortest path between the TLC of W_1 and the TRC of W_n.

Consider the path that has three widgets involved, namely W_i, W_j, W_k, where $i < j < k$, as shown in Figure 6.33, where the path is $A \to B \to C$ through two arc sequences AB and BC. Let L_1 denote the cost (length) of the arc sequence AB, and L_2 denote the cost (length) of the arc sequence BC, and let L_{XY} denote the cost of level 2 path from nodes X to Y, then we first note that $L_{AC} = L_{DC} - L_{DA}$. Also from the proof of "The Path" we know that that $L_{DC} \leq L_2 + c_{DB} = L_2 + 1$, therefore we have

$$L_{AC} = L_{DC} - L_{DA} \leq L_2 + 1 - L_{DA} \leq L_2 \leq L_2 + L_1,$$

and thus, we proved that the level 2 path from A to C is less than the union of the two arc sequences AB and BC combined.

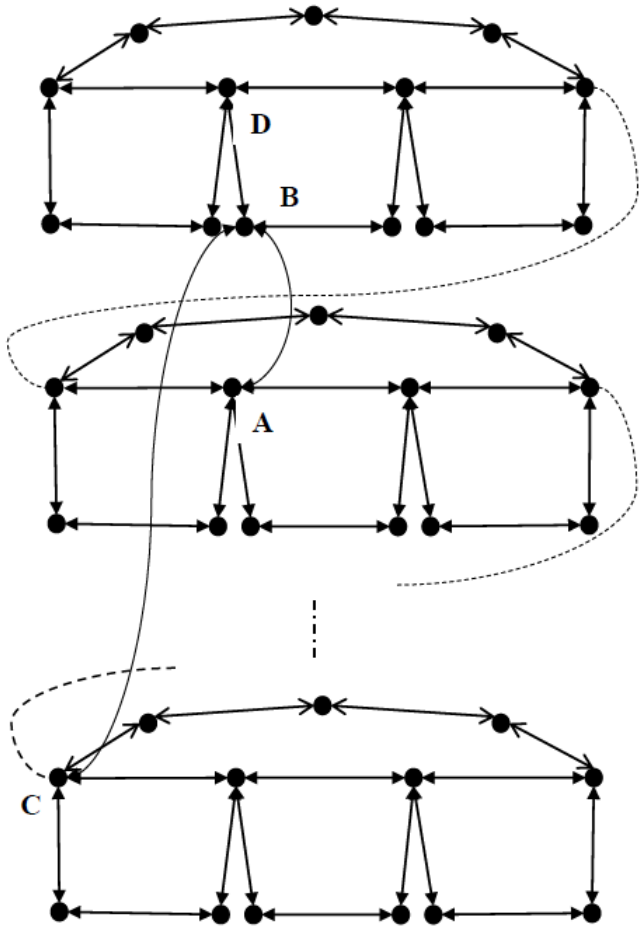

FIGURE 6.33: the path that goes through three widgets

Note that Type II paths cannot have more than 3 widgets involved, since from each

window, there are at most two other windows that are connected by the widget-linking arc sequences. Therefore, each window can go to at most two other windows without passing a level 2 node.

Thus, we've shown that any Type II path can be replaced by the Type I path with the same start and end node without incurring more costs.

Therefore, this case is verified.

6.3.1.2 The Roof

Since "the Path" is the shortest path between TLC of W_1 and TRC of W_n, by principle of optimality, we know that the all level 2 subpaths are shortest paths between their origin and destination, therefore, since TLC and TRC of each widget are level 2 nodes, we know that the level 2 path between TLC and TRC of each widget is the shortest path, with weight 4. Since the roof of each widget also has weight 4, we know that the roof is also the shortest path between TLC and TRC of any widget. Thus, this case is verified.

6.3.1.3 Within Window

Consider the bottom right corner of any window. If we want to go to the top left corner, one way is that we can stay within the window in two ways, one uses the level 1 arc and the other uses the level 2 arc. Since we set the weights so that the level 1 and level 2 arc of the same window has the same value, and also we set the vertical arc with weight 1, we can conclude that the two paths have the same weight, either 2 or 3. The other choice is to jump across to another widget, from the analysis in the roof section we know that this type of path has at least weight 4. Thus, any path of this kind will have weight at least 3. Therefore, this case is verified.

The only exception occurs when the two windows are adjacent and are across two widgets as follows:

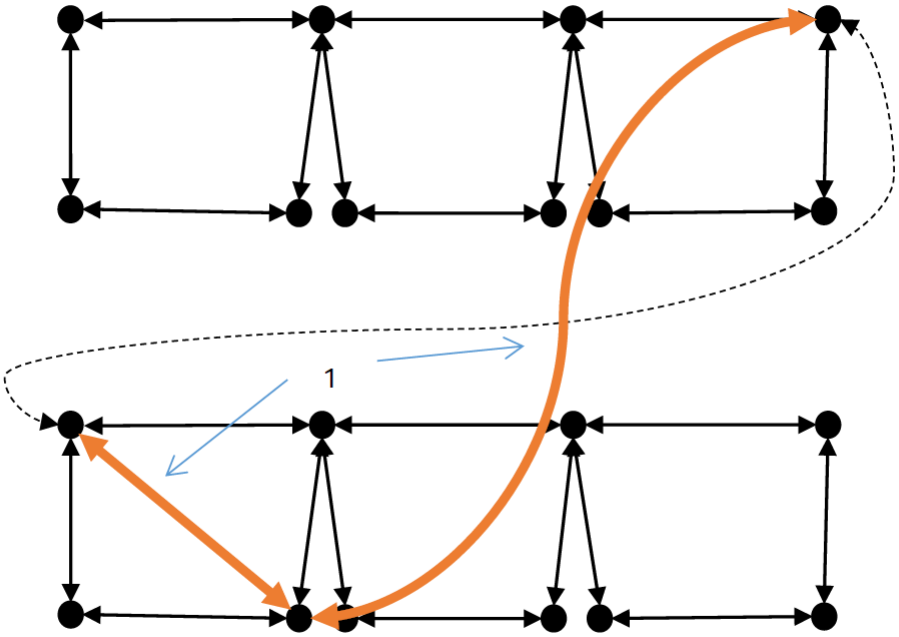

FIGURE 6.34: The Adjacent Cross-widget Windows

Note that the two thick arcs are in fact the same arc, since the two level 2 nodes are actually contracted, which makes the arc sequence (one arc) the short cut between the top left corner and the bottom right corner of the window. Because of this, this arc, with weight 1, becomes the shortcut between the top left node and the bottom right node of the window.

To avoid this, we only need to make sure the adjacent windows across widgets never represent the same sets. We avoid this by making sure RW of W_i and LW of W_{i+1} represent different sets, for all $i = 1, 2, \ldots, n-1$. We do this by going through each window in sequence and if two sets are adjacent, then we swap the latter set with its immediate neighbour. As sets are unique within a widget, we know we only have to do this once.

Cross Windows: For each two windows that are connected by the widget-linking arc sequences, we want to enforce the level 1 arc to be no longer than the level 2 arc, i.e., the shortest path between the TLC of W_i that contains the first window to the bottom right corner of W_j that contains the second window is the one uses the level 1 arc, and similarly, the shortest path between the bottom left corner of W_i and the TRC of W_j is the one uses the level 1 arc as well, which can be seen from

Figure 6.35: (the thick paths indicate the shortest path between the corresponding origin and destination, and the dashed ones are the alternatives).

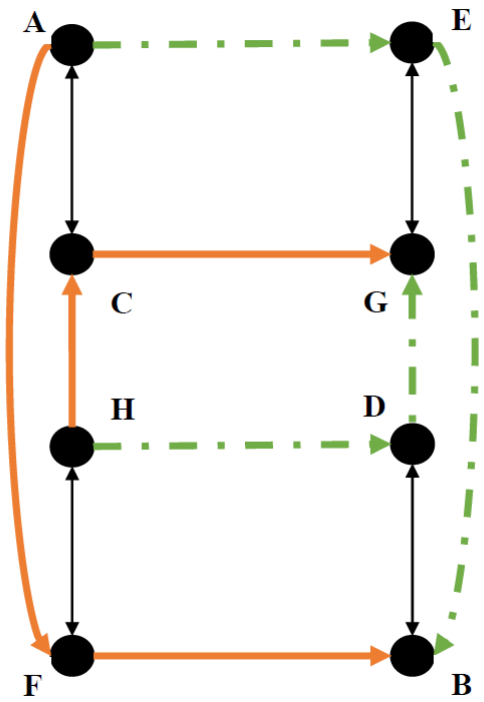

FIGURE 6.35: Illustration of the Cross-Widgets shortest paths

As we proved that with the weight setup at the beginning of this section, "The Path" is the shortest path, by principle of optimality, any subpath of "The Path", i.e., any path that consists of only the level 2 path, is the shortest path between its end nodes. For the rest of this proof, we denote $X \xrightarrow{\text{lvl2 path}} Y$ the shortest path from X to Y that uses level 2 arcs exclusively, i.e., the subpath of "The Path".

We consider the origin-destination pair (A, B) in the graph above. Below we enumerate the possible paths between the A, B, in order to show path $A \to F \to B$ is the shortest between A and B. Note that due to symmetry, the path $A \to F \to B$ is equivalent to the path $B \to F \to A$. Also note that from point B, we can either go to F, D or E.

- $B \to D$: Since the shortest path from D to A is the long level 2 path based on the principle of optimality. Therefore, the shortest path between AB in this case is $B \to D \xrightarrow{\text{lvl2 path}} A$.

- $B \to E$: The shortest path between E to A is the arc EA. So the shortest path is $B \to E \xrightarrow{\text{lvl2 path}} A$.

- $B \to F$: either $F \to A$ or $F \to H \xrightarrow{\text{lvl2 path}} A$. Note that the path $B \to F \to H \xrightarrow{\text{lvl2 path}} A$ has weight equals to $B \to D \xrightarrow{\text{lvl2 path}} A$; the path $B \to F \to A$ has weight equals to the path $B \to E \to A$, since the arc sequences AF and BE have the same number of arcs, and hence the same weight.

We define posdif(XY) be the position difference of two windows whose bottom left corner are X, Y respectively. Combine the three cases above, we only need to compare $A \to F \to B$ and $A \xrightarrow{\text{lvl2 path}} H \to D \to B$. Consider $A \to F \to B$, it has weight equals to the position difference between C and F, by the definition of the arc sequences, plus the weight of arc BF, denoted as $c(FB)$, i.e.,

$$\text{posdif}(CF) + c(FB).$$

On the other hand, we first prove the following proposition:

Proposition 6.13. *Consider two level 2 nodes A, H, $A \xrightarrow{\text{lvl2 path}} H$ have weight at least the position difference between C and F minus 1, where C, F are two level 1 nodes joined with A, H, respectively, as shown in Figure 6.35 i.e.,* posdif$(CF) - 1$.

Proof. Consider the level 2 subpath from A to H, $A \xrightarrow{\text{lvl2 path}} H$. Assume that A is in widget $W1$ and H is in widget $W2$, and there are $m-1$ widgets in between $W1$ and $W2$. We cut the path $A \xrightarrow{\text{lvl2 path}} H$ into three pieces: $P1 = A \xrightarrow{\text{lvl2 path}} H \cap W1$ (the level 2 subpath from A that is contained in $W1$), $P2 = A \xrightarrow{\text{lvl2 path}} H \cap W2$ (the level 2 subpath to H that is contained in $W2$), and the level 2 subpath in the widgets between $W1$ and $W2$, namely $P3$. Notice that with the current weight setting, the weight of $P3$ is fixed and equals to $4(m-1)$.

Consider the five cases where the position difference was defined, and each of the following cases corresponds to Figure 6.10 (Case 1), Figure 6.11 (Case 2), Figure 6.12 (Case 3), Figure 6.13 (Case 4), Figure 6.14 (Case 5),

Case1: There are three arcs in $P1 \cup P2$, and each of them can have weight as small as 1, so $\lambda_{P1} + \lambda_{P2} + \lambda_{P3} = 4(m-1) + 3 \times 1 = 4m - 1 = \text{posdif}(CF) - 1$,

Case2: There are two arcs in $P1 \cup P2$, and each of them can have weight as small as 1, so $\lambda_{P1} + \lambda_{P2} + \lambda_{P3} = 4(m-1) + 2 \times 1 = 4m - 2 \geq (4m-2) - 1 = \text{posdif}(CF) - 1$,

Case3: There are one arcs in $P1 \cup P2$, and each of them can have weight as small as 1, so $\lambda_{P1} + \lambda_{P2} + \lambda_{P3} = 4(m-1) + 1 \times 1 = 4(m-1) + 1 = 4m - 3 \geq (4m-3) - 1 = \text{posdif}(CF) - 1$,

Case4: There are four arcs in $P1 \cup P2$, and each of them can have weight as small as 1, so $\lambda_{P1} + \lambda_{P2} + \lambda_{P3} = 4(m-1) + 4 \times 1 = 4(m-1) + 4 = 4m = (4m+1) - 1 = \text{posdif}(CF) - 1$,

Case5: There are five arcs in $P1 \cup P2$, and each of them can have weight as small as 1, so $\lambda_{P1} + \lambda_{P2} + \lambda_{P3} = 4(m-1) + 5 \times 1 = 4(m-1) + 5 = 4m + 1 = (4m+2) - 1 = \text{posdif}(CF) - 1$.

As we enumerate all the possibilities of position differences, and we can conclude that the weight of $A \xrightarrow{\text{lvl2 path}} H$ is at least posdif(CF) $-$ 1. $\square$

$H \to D \to B$ has weight equals to the weight of arc FB plus 1 (weight of arc DB). Therefore, the weight of $A \xrightarrow{\text{lvl2 path}} H \to D \to B$ is at least

$$\text{posdif}(CF) - 1 + c(FB) + 1 = \text{posdif}(CF) + c(FB),$$

which equals to the weight of $A \to F \to B$. Therefore, $A \to F \to B$ is the shortest path between A and B. Similar argument can be applied to the origin-destination pair C, D by comparing the weight of $C \to G \to D$ and $C \to A \xrightarrow{\text{lvl2 path}} D$.

Therefore, this case is verified.

Now, we have verified all the prescribed shortest paths that are the shortest under the weights. For all other pairs of nodes, we have several cases:

- Both nodes are on the same widget-linking arc sequence: we will choose just the path on the arc sequence connecting the two nodes. It is the shortest path because any arc sequence is the shortest path, and the principle of optimality implies the result.

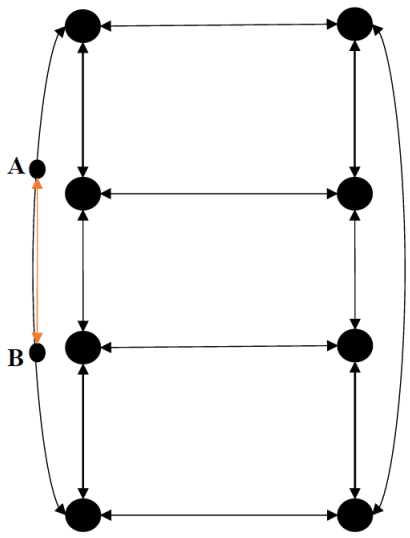

FIGURE 6.36: Path from A to B in the same widget-linking arc sequence

- Both nodes are part of the "attached chain": we set the path on the chain connecting the two nodes to be the shortest, and clearly it is the shortest.

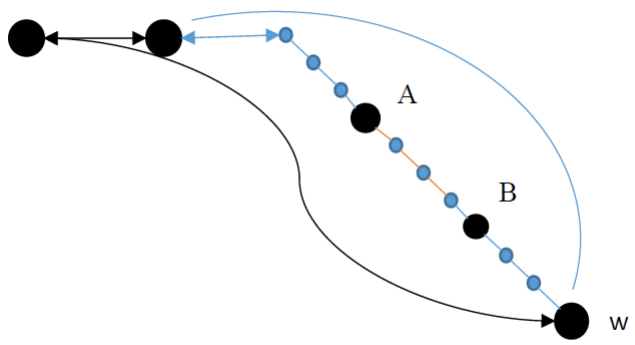

FIGURE 6.37: Path from A to B in the same chain.

- One node is in an "attached chain": we set the shortest path to be the shortest path from the other node to the root of the chain union a subset of the chain to reach the target node. This type of paths are shortest because for any node in the chain, if it tries to reach any other node outside of this chain, it has to pass through the root of the chain first.

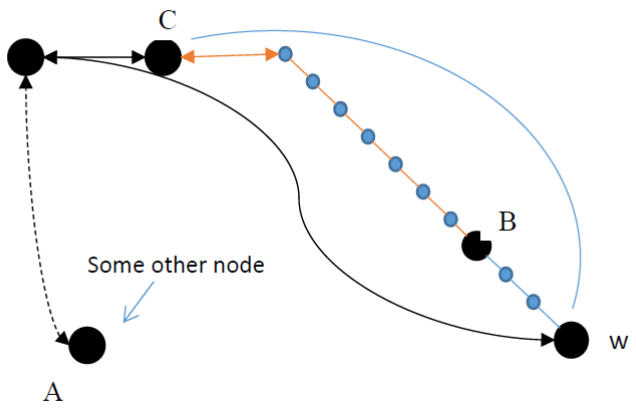

FIGURE 6.38: Path from A to B crossing C as a root of a chain containing B

- One node is in the tail chain: we set the shortest path to be the shortest path from the other node to the top right corner of the last widget union a subset of the chain to reach the target node. Similar reason from last case applies.
- All the other cases (nodes on the widgets and on the widget-linking arc sequences): Note that we added an arc to connect the pair of nodes and let these single arcs be the shortest paths between its two adjacent nodes. Each of these arcs can be assigned with the weight exactly equal to the weight of the shortest path of the corresponding start and end nodes upon the removal of the arc itself, and all the properties of the widgets remains unchanged.

6.3.2 Verifying the Objective Value

At last, we will show why the objective is indeed $26n$. In fact, since the arc with largest cost is the arc that connects the TLC of $W1$ and the end of the tail chain, which has cost $26n$, it is sufficient to show that any arc has weight at most $26n$. The only other arcs that can have weights more than 2 are the ones added at the end whose cost equals to the cost of the shortest path between its two end nodes. Note that the weight of these arcs equal to the weight of the shortest path bewteen the two end nodes if we remove the arc. To analyze the weight of these extra arcs, we can in fact remove all of them and prove that in the remainder graph, the shortest path between any two nodes in "The Widgets" but the attached chains and the tail

chain, has cost at most $26n$. The subgraph includes arcs only from the original widget structure, i.e., the windows and the roofs, plus all the widget-linking arc sequences.

In fact, consider any two nodes on the windows or the roofs. Since the total number of arcs on the widgets structure (windows plus the roofs) are $16n$, and for each widget, there are exactly two arcs with weight 2, and every other arc has weight 1, so the sum of the weight of all the arcs on windows and roofs are $18n$, and thus any path between this pair of nodes can have weight at most $18n$. On the other hand, we know that the largest possible arc sequence is the one that connects the LW of W_1 and the RW of W_n, which has weight at most $4n$. Therefore, for any two points, the shortest path must not exceed $18n + 4n + 4n = 26n$.

Therefore, we've shown that the objective is $26n$.

Thus, given a partitionable instance, the corresponding routing problem is feasible and the optimal value is $26n$.

This completes the reduction.

Chapter 7

Conclusion

In this dissertation, we went through the Internet routing problem where traffic is congested, and saw approaches developed in practice, such as Random Early Detection (RED). Motivated by all the previous work, we modelled the OSPF routing problem under RED as a Integer non-linear math program, which found a routing policy that attempts to optimize the total flow delivered when network is operated under RED. and showed that the program is non-convex. We also introduced the Inverse Shortest Path problem that we used to convert the derived policy into OSPF arc weights. We considered the All-Pair variants of the OSPF routing problem under RED and the Inverse Shortest Path problem and we proved that both of these problems are NP-Hard.

We conducted experiments to test the performance of our model using traffic data from 3 real-world research networks. We used the corresponding industry standard OSPF policies as benchmarks. Using an off-the-shelf solver, Bonmin, we obtained local solutions for all hourly instance of 3 networks, and did a comparison of the total flow received between the derived policy and the benchmark. We showed that our policy outperformed the industry benchmark in general. In addition, we use a min-max approach to develop a robust model that produces one policy that takes into account one full week of traffic demands. The experiments for the robust model show that we can obtain a robust policy from the solver; at the same time, the robust policy outperformed the industry standard as well. Moreover, we tested whether the generated path from our models were realizable, and the results show that every policy was OSPF-configurable.

7.1 Future Work

First, in this thesis we focused exclusively on the OSPF routing protocol, or shortest path protocols in general. Specifically, our model derived one simple path for each origin-destination pair, meaning that all the flow must be sent through a unique node-disjoint path. However, there are other protocols in practice that do not have this constraint. Commodities under these protocols may be sent over multiple paths. For example, equal-cost multipath routing is based on the shortest path algorithm but equally split the flow over all equally weighted paths, i.e., there may be multiple shortest paths for each origin-destination pair. These routing problem were also considered in the Traffic Engineering problem, but few have attempted to incorporate Active Queue Management into the formulation.

Second, we showed that the All-Pair Inverse Shortest Path Problem is NP-Hard; however, it should be noted that we assumed that the prescribed shortest paths were not necessarily the unique shortest path. Our proof was inspired by Bley's work; while in his work, he assumed that the paths were uniquely the shortest. It is still open whether it is hard to solve the All-Pair Inverse Shortest Path Problem where paths must be unique shortest paths.

Finally, our experiments showed that all the generated policy of our model is OSPF-configurable, and more importantly, the weights were mostly in the order of 10. This was an interesting observation since in the Abilene Network and CANARIE, the arc metrics were in the order of 10^4. The necessary conditions for a set of paths to be OSPF realizable were discussed [10]. In our setting, the all-pairs assumption implies that the subgraph induced by the paths from the same origin must form a shortest path tree, providing a necessary condition that the non-all-pairs counterpart was not able to incorporate. It is yet to be shown why all the instances were realizable and why all the arc weights are significantly smaller than the industry standard.

Appendix A

Ampl Code

A.1 Model III

```
param K;
# THE COMMODITY INDEX
set Commodities := 1..K;
param V;
# THE VERTEX INDEX
set Vertices := 1..V;

# THE EDGE SET
set Edge within{Vertices, Vertices};

param beta;
param cap {Edge};

param M := 10;

# THE SOURCE QUANTITY
param Source {Commodities};
# THE COST VALUE
param Cost {Commodities};

# THE FOLLOWING THREE SETS IS USED TO SPECIFY THE GRAPH
```

```
# AND THE CORRESPONDING OD PAIR; MORE DETAILED INFO
# WOULD BE SPECIFIED IN THE DAT FILE
# THE ORIGIN SET
set Orig within {Vertices, Commodities};
# THE DESTINATION SET
set Dest within {Vertices, Commodities};

# THE COMMODITY PAIR REPRESENTING ALL O-D D-O
set CommPair within {Commodities, Commodities};

#####################################################################
# the Variable x representing the received flow of Commodity k
# at each node i
# the x_i,k in the paper
#####################################################################
var x {Vertices, Commodities} >= 0;

#####################################################################
# the Variable representing the proportion of flow of Commodity k
# sending from node i to node j; here for each commodity there will
# be only one destination node from each node
# i.e. Binary variable subject to that sum = 1
# the alpha_i,j,k in the paper
#####################################################################
var portion {Edge, Commodities} binary;

#####################################################################
# the Variable representing the proportion of flow of Commodity k
# sending from root i; here for each commodity there will
# be only one destination node from each node
# i.e. Binary variable subject to that sum = 1
# the alpha_i,j,u in the paper
#####################################################################
var portiontotal {Edge, Vertices} binary;

#####################################################################
```

```
# a intermediate variable representing all the flow received at node j
# (could be multicommodity node)
# corresponding to the summation in the RED constraints in the paper
#####################################################################
var total {Edge};

#####################################################################
# an intermediate variable to specify the gain function f,
# for each arc
# corresponding to the gain function f in the paper
#####################################################################
var f {Edge};

# OBJECTIVE: MAXIMIZE THE TOTAL AMOUNT OF WEIGHTED FLOW AT DESTINATION
maximize WEIGHTEDFLOW:
        sum {(i,k) in Dest} Cost[k]*x[i,k];

#####################################################################
# Origin Condition Constraint
# x_ok,k = s_k in the paper
#####################################################################
subject to OrigCond {(i,k) in Orig}:
        x[i,k] = Source[k];

#####################################################################
# the new Conservation Constraints
#####################################################################
subject to SourConserv1 {k in Commodities}:
        sum {(i,k) in Orig, (i,j) in Edge} portion[i,j,k] = 1;

subject to SourConserv2 {k in Commodities}:
        sum {(i,k) in Orig, (j,i) in Edge} portion[j,i,k] = 0;

subject to DestConserv1 {k in Commodities}:
        sum {(i,k) in Dest, (j,i) in Edge} portion[j,i,k] = 1;
```

```
subject to DestConserv2 {k in Commodities}:
        sum {(i,k) in Dest, (i,j) in Edge} portion[i,j,k] = 0;

subject to BalanceConserv0 {(k,l) in CommPair, (u,v) in Edge}:
        portion[u,v,k] = portion[v,u,l];

subject to BC {(j,k) in {Vertices, Commodities} diff (Dest union Orig)}:
        sum{(i,j) in Edge}portion[i,j,k]=sum{(j,i) in Edge}portion[j,i,k];

subject to OPC {(j,k) in {Vertices, Commodities} diff (Dest union Orig)}:
        sum {(i,j) in Edge} portion[i,j,k] <= 1;

subject to TreeConstraint {u in Vertices}:
        sum {(i,j) in Edge} portiontotal[i,j,u] = V-1;
subject to RootSubGraphSingle {(u,k) in Orig, (i,j) in Edge}:
         portion[i,j,k] <= portiontotal[i,j,u];

#####################################################################
# the Main flow constraints, Random Early Detection
# The flor received at node j for commodity k is a sum over the flow
# sent from all arcs and multiply by the RED factor
# sum_i,j alpha_i,j,k * x_i,k * f(sum_k alpha_i,j,k * x_i,k) = x_j,k
#####################################################################
subject to RED {(j,k) in {Vertices, Commodities} diff Orig}:
        sum {(i,j) in Edge} portion[i,j,k]*x[i,k]*f[i,j] = x[j,k];

#####################################################################
# Constraints worked for the helper variables
#####################################################################
subject to TotalFlowIn {(i,j) in Edge}:
        total[i,j] = sum {k in Commodities} portion[i,j,k]*x[i,k];

subject to Approximate {(i,j) in Edge}:
        f[i,j] = cap[i,j]/(cap[i,j]+total[i,j]);
```

A.2 ISP Model

```
param K;
set V := 1..K;
set Edge within {V,V};
set Path within {V,Edge};

var y {V,V};
var c {Edge} integer >= 0;

maximize void: 0;

subject to Opt {(u,w,v) in Path}:
    y[u,v] - y[u,w] = c[w,v];

subject to Err {(u,w,v) in {V,Edge} diff Path}:
    y[u,v] - y[u,w] <= c[w,v] - 1;

subject to Pos {(v,w) in Edge}:
    c[v,w] >= 1;

subject to Bal {(v,w) in Edge}:
    c[v,w] = c[w,v];

subject to Zeros {u in V}:
    y[u,u] = 0;
```

Appendix B

Sample Policies

B.1 Abilene Network

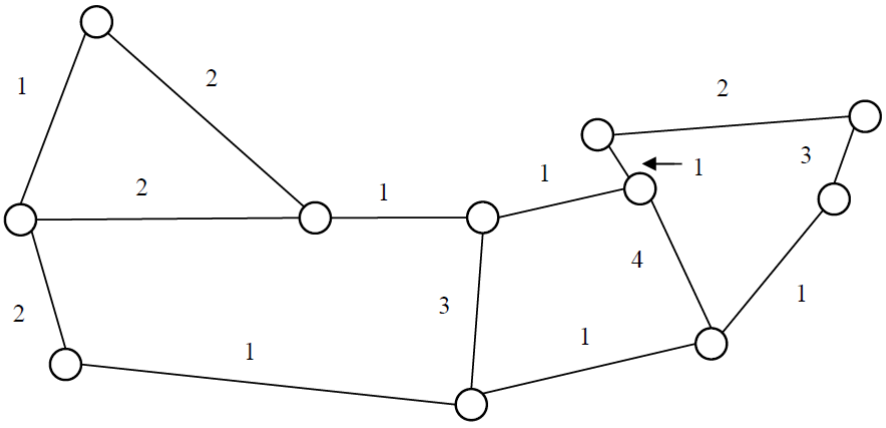

FIGURE B.1: Sample policy for Abilene Network

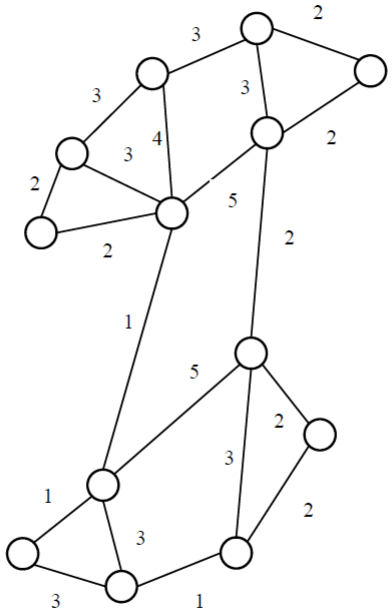

FIGURE B.2: Sample policy for TWAREN

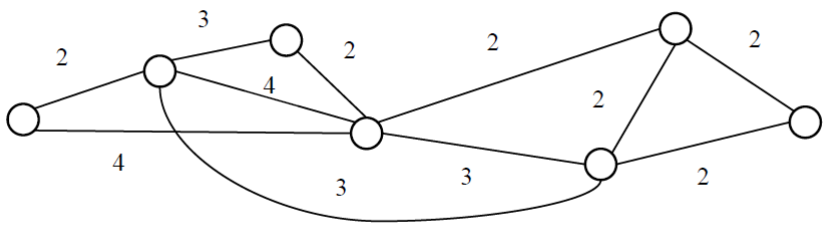

FIGURE B.3: Sample policy for CANARIE

Bibliography

[1] Bonmin Home Page: Basic Open-source Nonlinear Mixed INteger Programming. http://www.coin-or.org/Bonmin/.

[2] CANARIE Network Operation Center. http://www.canarie.ca/en/network/overview/.

[3] Class-Based Weighted Fair Queueing and Weighted Random Early Detection. http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfconav.pdf.

[4] Release 12.1 Quality of Service Solutions Configuration Guide—Congestion Avoidance Overview. http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfconav.pdf.

[5] TWAREN Network Operation Center. http://noc.twaren.net/noc_eng/index.php/.

[6] (1997). Configuraing OSPF. http://www.cisco.com/c/en/us/td/docs/security/asa/asa82/configuration/guide/config/route_ospf.pdf.

[7] (2011). Cisco Visual Networking Index: Global Mobile data Traffic Forecast Update 2009-2014. White Paper.

[8] (2014). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018. White Paper.

[9] Alazemi, H. M., Mokhtar, A., and Azizoglu, M. (2001). Stochastic Approach for Modeling Random Early Detection Gateways in TCP/IP Networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 8, pages 2385–2390. IEEE.

[10] Altın, A., Fortz, B., Thorup, M., and Ümit, H. (2009). Intra-domain Traffic Engineering with Shortest Path Routing Protocols. *4OR*, 7(4):301–335.

[11] Awduche, D., Chiu, A., Elwalid, A., Widjaja, I., and Xiao, X. (2002). Overview and Principles of Internet Traffic Engineering. RFC 3272 (Informational). Updated by RFC 5462.

[12] Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., and McManus, J. (1999). Requirements for Traffic Engineering Over MPLS. RFC 2702 (Informational).

[13] Aweya, J., Ouellette, M., and Montuno, D. Y. (2001). A Control Theoretic Approach to Active Queue Management. *Computer Networks*, 36(2):203–235.

[14] Bley, A. (2007a). Inapproximability Results for the Inverse Shortest Paths Problem with Integer Lengths and Unique Shortest Paths. *Networks*, 50(1):29–36.

[15] Bley, A. (2007b). *Routing and Capacity Optimization for IP Networks*. PhD thesis, TU Berlin.

[16] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and Zhang, L. (1998). Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309 (Informational). Updated by RFC 7141.

[17] Burton, D. and Toint, P. L. (1992). On an Instance of the Inverse Shortest Paths Problem. *Mathematical Programming*, 53(1-3):45–61.

[18] Comer, D. and Stevens, D. L. (2003). *Internetworking with TCP/IP*. Prentice-Hall.

[19] Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271.

[20] Dimitrov, S., Epelman, M., and Sharma, D. (2009). New Models of Network Routing under Active Congestion Control. *Michigan University, Industrial and Operations Engineering Department*.

[21] Even, S., Itai, A., and Shamir, A. (1975). On the Complexity of Time Table and Multi-commodity Flow Problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 184–193. IEEE Computer Society.

[22] Feldmann, A. and Rexford, J. (2001). IP Network Configuration for Intradomain Traffic Engineering. *Network, IEEE*, 15(5):46–57.

[23] Floyd, S. (2000). Congestion Control Principles. RFC 2914 (Best Current Practice). Updated by RFC 7141.

[24] FLOYD, S. (2001). Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. *Technical Report*.

[25] Floyd, S. and Jacobson, V. (1993). Random Early Detection Gateways for Congestion Avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413.

[26] Fortz, B. and Thorup, M. (2000). Internet Traffic Engineering by Optimizing OSPF Weights. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, volume 2, pages 519–528. IEEE.

[27] Fortz, B. and Thorup, M. (2006). Optimizing ospf/is-is weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767.

[28] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*.

[29] Gonzalez, T. F. (1985). Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306.

[30] Hawkinson, J. and Bates, T. (1996). Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930 (Best Current Practice). Updated by RFCs 6996, 7300.

[31] Kurose, J. F. and Ross, K. W. (2001). *Computer Networking: A top-down Approach Featuring the Internet*, volume 2. Addison-Wesley Reading.

[32] Moy, J. (1998). OSPF Version 2. RFC 2328 (INTERNET STANDARD). Updated by RFCs 5709, 6549, 6845, 6860.

[33] Nagle, J. (1984). Congestion Control in IP/TCP Internetworks. *SIGCOMM Comput. Commun. Rev.*, 14(4):11–17.

[34] Rekhter, Y., Li, T., and Hares, S. (2006). A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard). Updated by RFCs 6286, 6608, 6793.