ICTEAM, pôle d'Ingénierie Informatique
Ecole polytechnique de Louvain
Université catholique de Louvain
Louvain-la-Neuve, Belgium

Traffic engineering techniques for data center networks

Trong-Viet HO

October 2012

Thèse présentée en vue de
l'obtention du grade de Docteur en
Sciences de l'Ingénieur

ABSTRACT

Traffic engineering consists in improving the performance of the telecommunication networks which is evaluated by a large number of criteria. The ultimate objective is to avoid congestion in the network by keeping its links from being overloaded. In large Ethernet networks with thousands of servers, such as data centers, improving the performance of the traditional switching protocols is a crucial but very challenging task due to an exploration in the size of solution space and the complexity. Thus, exact methods are inappropriate even for reasonable size networks.

Local Search (LS) is a powerful method for solving computational optimization problems such as the Vertex Cover, Traveling Salesman, or Boolean Satisfiability. The advantage of LS for these problems is its ability to find an intelligent path from a low quality solution to a high quality one in a huge search space. In this thesis, we propose different approximate methods based on Local Search for solving the class of traffic engineering problems in data center networks that implement Spanning Tree Protocol and Multiple Spanning Tree Protocol.

First, we tackle the minimization of the maximal link utilization in the Ethernet networks with one spanning tree. Next, we cope with data center networks containing many spanning trees. We then deal with the minimization of service disruption and the worst-case maximal link utilization in data center networks with many spanning trees. Last, we develop a novel design of multi-objective algorithms for solving the traffic engineering problems in large data centers by taking into account three objectives to be minimized: maximal link utilization, network total load and number of used links.

Our schemes reduce significantly the size of the search space by releasing the dependence of the solutions from the link cost computation in order to obtain an intended spanning tree. We propose efficient incremental techniques to speed up the computation of objective values. Furthermore, our approaches show good results on the credible data sets and are evaluated by the strong assessment methods.

ACKNOWLEDGEMENTS

First of all, I would like to thank Yves Deville and Olivier Bonaventure for the four years they spent helping me. I am very lucky to have two great advisors of two different promoting styles. Yves Deville is a very demanding supervisor who wants every smallest detail to be clear and intuitive. His ideas and his criticisms sometimes made me nervous or even angry but I could not deny the truth that they were the best advices. As an expert in the network domain, Olivier Bonaventure gave me the initial ideas and the background for almost all of my research subjects. He spent all his time and his energy for his students. I learned a lot from their honesty and their responsibility in both research and life style.

I thank also Pierre Francois for his ideas and his supports that helped me a lot in the first and difficult period of my PhD. Besides his contributions to this thesis, he is also one of my close friends at UCL.

Many thanks goes to Quang Dung Pham who has shared the same office with me for four years. He was like my brother in the BeCool team. He supported me a lot in research and daily life in Louvain-la-Neuve.

I want to warmly thank my other teammates in the BeCool group (Stéphane Zampelli, Pierre Schaus, Jean-Noël Monette, Sébastien Mouthuy, Julien Dupuis, Vianney le Clément, Florence Massen, Jean-Baptiste Mairy, and Quoc Trung Bui) for their ideas and constructive comments to this thesis. I will never forget these fruitful years of working with them - my second family.

Thanks goes as well to the whole INGI department and staff who organized frequently the different research and social activities that made working at UCL interesting and friendly.

I give my thanks to the member of the jury who accepted to read and to comment this thesis. Especially Bernard Fortz that gave me the precious idea of multi-criteria for traffic engineering.

I thank all my Vietnamese friends in Louvain-la-Neuve for including me in a special community that gave me an interesting life out of research with different sport and cultural activities. They helped me a lot to relieve the nostalgia of my country - Vietnam.

The thanks cannot express my gratitude to my parents who raised me up and encouraged me in my whole life. They are really great examples for me. Thank you parents for everthing that you do for me!

Last but not least, I would like to thank my wife who is always there to support and to encourage me in every difficult moment of my life. I am very lucky to have her in my life.

This research has been supported by the Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture (FRIA).

TABLE OF CONTENTS

Table of Contents **vii**

1 Introduction **1**

1.1 Traffic engineering in switched Ethernet networks . . . 1
1.2 Challenges 2
1.3 Local Search 3
1.4 Contributions 3
1.5 Outline 5

2 Background and Related Work **7**

2.1 Data Centers and Switching Protocol variants 7
2.1.1 Spanning Tree Protocol 802.1d 9
2.1.2 Multiple Spanning Tree Protocol 802.1s 10
2.2 Traffic Engineering in Data Center Networks 12
2.2.1 Maximal link utilization 13
2.2.2 Total load 15
2.2.3 Number of used links 16
2.2.4 Service disruption after link failures 17
2.3 Constraint Satisfaction Problem 19
2.4 Local Search 20
2.4.1 Principles 20
2.4.2 Metaheuristics 21
2.5 Multi-objective Optimization 23
2.5.1 Pareto dominance and Pareto front 23
2.5.2 Multi-objective Optimization Methods 25
2.5.3 Local Search for Multi-objective Optimization 26

3 Ethernet networks using Spanning Tree 29
3.1 Introduction 29
3.2 Problem Formulation 31
3.3 Spanning Tree Optimization using Local Search 33
3.3.1 Algorithm Description 33
3.3.2 Spanning Tree Modeling and Query 34
3.3.3 Root Selection 35
3.3.4 Neighborhood Formulation 36
3.3.5 Magic Cycle for Link Load Update 40
3.4 Link Cost Generation 41
3.5 Data Generation 42
3.5.1 Generic topologies 42
3.5.2 Portland 46
3.5.3 Fat Tree 47
3.5.4 Traffic Demand Matrix 48
3.6 Experiments and Results 50
3.6.1 Improvement compared to STP 802.1d default solution 51
3.6.2 Comparison with IGP Weight Optimization . . 53
3.7 Conclusion 55

4 Joint optimization of Multiple Spanning Trees 57
4.1 Introduction 57
4.2 Problem Description 60
4.3 MSTP Optimization Using Local Search 61
4.3.1 Algorithm description 61
4.3.2 Removing an edge 63
4.3.3 Adding an edge 64
4.3.4 Speeding up the search 64
4.4 Data Generation 65
4.4.1 Data Center topologies 65
4.4.2 Traffic demand matrices and VLANs generation 67
4.5 Experiments and Evaluation 70
4.6 Conclusion 73

5 Dealing with link failures 81
5.1 Introduction 81
5.2 Problem Statement 83
5.3 Heuristic Algorithm for MSTP in case of link failures . 85
5.4 Dependence of Back-up links on Link Cost Configuration 87
5.5 Link Cost Update Technique 90
5.6 Data Set Composition 92
5.7 Experiments and Results 93
5.8 Conclusion 95

6 Multi-objective Traffic Engineering 99
6.1 Introduction 99
6.2 Problem Statement 101
6.3 Proposed Algorithms 103
6.3.1 Single-objective Approaches 103
6.3.2 Bi-objective Optimization 106
6.4 Performance Comparisons 109
6.4.1 Assessment Metrics for Multi-objective Algorithms 110
6.4.2 Data Set Composition 115
6.4.3 Results and Evaluation 115
6.5 Conclusion 118

7 Conclusion 123
7.1 Results 123
7.2 Future Work 125

A Cycle Load Update Proof 127

B Link Cost Update Proof 133

C Multi-objective TE Result Details 135

Bibliography 139

1 INTRODUCTION

The aim of this thesis is to provide traffic engineering techniques for optimizing the performance of the large-scale Ethernet networks that implement Spanning Tree Protocol (STP) or Multiple Spaning Tree Protocol (MSTP).

1.1 Traffic engineering in switched Ethernet networks

Traffic engineering (TE) is known as an optimization method for improving the performance of a telecomunication network by analyzing, forecasting and efficiently adjusting the traffic (data) transmission in the network. TE is an essential task in all kinds of telecomunication networks, including the Public Switched Telephone Networks (PSTN), Local Area Networks (LAN), Ethernet networks, Wide Area Networks (WAN), Cellular Telephone networks and the Internet networks [NRR11].

Data centers with huge logical storage capacity are now base of many Internet and cloud computing services. Deploying by switched Ethernet networks, data centers are mostly dominated by different variants of Spanning Tree Protocol. TE problems in Data Center Networks (DCN) are very varied, both in problem formulations and evaluation criteria (objectives). Some typical problem formulations for TE in DCNs are off-line TE, online TE, Virtual LAN (VLAN) to spanning tree mapping, traffic flow to spanning tree assignment, and multiple spanning tree construction. The ultimate objective of a TE method is to avoid the congestion in the DCN. Thus, a minimization of the maximal link utilization value is often the most important evaluation criterion. Other common TE objectives that can be taken into account are link delay, network total load, energy saving, number of used links, and service disruption in case of failures. More details are presented in Chapter 2.

This research focuses on the design and evaluation of TE techniques to solve four types of problems:

- Minimization of the maximal link utilization in the Ethernet networks using spanning tree.
- Minimization of the maximal link utilization in the DCNs with multiple spanning trees.
- Minimization of service disruption and the worst-case maximal link utilization in the DCNs with multiple spanning trees in case of link failures.
- Multi-objective TE for DCNs with multiple spanning trees. For this problem, we take into account three objectives to be minimized: maximal link utilization, network total load and number of used links.

1.2 Challenges

Most of TE problems are NP-hard with an exponential solution space. TE in typical IP networks with less than 100 nodes (routers) has been a very challenging problem in the last decade [For00]. Current powerful data centers contain thousands of servers and hundreds of switches that are divided into many VLANs [BAM10, BAAZ10]. TE methods for DCNs deal with an explosion in the size of solution space and the complexity. In practice, network operators would like to provide the decisions as fast as possible. This means that exact methods are not appropriate even for reasonable size DCN instances. The second challenge is to define TE techniques for DCNs when several objectives are required to be optimized. A single computation for each objective value is already costly. Other challenges are to cope with the online optimization problems where the traffic demands are continuously varied, to deal with the online reconfiguration of DCN because of modifications, etc... Thus, the design of TE approaches requires many efficient techniques to fit the practical demands of current DCNs.

1.3 Local Search

Approaches for solving TE problems in DCNs [Med06, HZC06, LYD$^+$03, SdSA$^+$09, dSS07, SdSA$^+$11, CJZ06, MSS09, PNM$^+$05] are often Stochastic methods, including Local Search, Meta-heuristic, Evolutionary Algorithm and many other Randomized Search Methods. The solutions obtained with these approximate approaches are often not optimal but acceptable when dealing with large DCNs.

Local search (LS) is a powerful technique that can find high quality solutions for computationally complex optimization problems (Vertex Cover, Traveling Salesman or Boolean Satisfiability) in polynomial time. LS can choose an interesting set of solutions to be explored from a huge search space by performing intelligent moves. LS starts with an initial solution and improves its quality by iteratively moving from one solution to one of the neighbor solutions. The search process finishes when some termination criterion is satisfied. A LS algorithm relies on a neighborhood function that defines the set of neighbors for each solution. A LS algorithm also uses a heuristic (such as best improvement, first improvement, random, etc...) to choose the neighbor. It can also use meta-heuristics, such as Tabu Search, Simulated Annealing, Variable Neighborhood Search to guide the search. Hence, local search algorithms are well-suited for solving the TE problems in large DCNs.

1.4 Contributions

The contributions of this thesis are the following:

1. We develop and implement an efficient local search algorithm for solving the TE problem of minimization of the maximal link utilization in Ethernet networks containing one spanning tree.

2. We extend our local search approach for networks containing one spanning tree to cope with the minimization of the maximal link utilization in DCNs containing many VLANs (one spanning tree per VLAN). Our local search scheme shows good performance for large DCNs.

3. We propose a heuristic algorithm for minimizing the worst-case maximal link utilization and the service disruption in case of link failures for large DCNs. The solutions obtained with our heuristic algorithm ensure a minimum service disruption and minimize the worst-case maximal link utilization after link failures.

4. We develop a novel design of multi-objective algorithms for solving the TE problem in large DCNs. Our multi-objective algorithms show good performance by integrating the heuristics of each single objective into the search process.

5. We introduce a solution space for the class of TE problems in DCNs with spanning trees. Our choice of doing the search directly on spanning tree space instead of the link cost space reduces significantly the size of the search space. This definition of search space also simplifies the definition of neighbor solutions in our LS algorithms.

6. We propose incremental algorithms to maintain the link loads and the all-pairs path lengths in the spanning tree. We show that the incrementality improves both complexity and efficiency in the design and the implementation of our LS algorithms.

7. There are few existing assessment methods for evaluating the performance of multi-objective TE techniques in DCNs. State-of-the-art approaches often evaluate their outcomes based on each single objective observation. We introduce a strong assessment method using different specific metrics for evaluating the multi-objective overall performance.

1.5 Outline

The remainder of the thesis is organized as follows:

Chapter 2 presents the background and the related work. Basic concepts of Ethernet and DCNs are introduced, followed by an overview of TE in DCNs, Local Search and Multi-objective Optimization. We also present the existing works in the domain.

Chapter 3 faces the challenge of designing and implementing an efficient LS algorithm for solving the TE problem in large-scale Ethernet networks using spanning tree. Maximal link utilization is taken into account as the objective to be minimized. We present how a network problem is modeled as a graph problem and how it can be solved with a local search algorithm. Methods for generating different network topologies and traffic demand matrices are also developed. We then evaluate the performance of our LS algorithm on the generated data sets.

Chapter 4 deals with the minimization of the maximal link utilization in the DCNs with multiple spanning trees. The TE problem in Ethernet networks is more challenging in large DCNs with more switches and many spanning trees are required to be generated. We analyze different studies on DCNs to obtain data sets that are representative of data centers. The description of our LS algorithm is presented, followed by a performance evaluation.

In Chapter 5, we propose a heuristic algorithm to ensure a good load balancing for the DCNs with a minimum service disruption in case of link failures. We show how to configure the link costs such that the MSTP protocol performs one unique link replacement when one of the links of the spanning tree links fails. This minimum service disruption is considered as a hard constraint in our heuristic algorithm for minimizing the worst-case maximal link utilization in the DCNs.

The design of multi-objective algorithms for solving the TE problem in DCNs is described in Chapter 6, we show how the heuristics of two single objective LSs can be included into the search process of a bi-objective LS. We also describe how we represent, update and evaluate the outcomes of a bi-objective algorithm. Different assessment metrics for multi-objective algorithms are adapted for evaluating the overall performance of our LS approaches.

Finally, a general conclusion and perspectives for future work are presented in Chapter 7.

2 Background and Related Work

Traffic engineering (TE) in Ethernet networks, including data centers is a widely researched topic because of both technology and performance issues. Recently, with the explosion of cloud computing services, TE in large DCNs becomes a critical need of many service providers. Different techniques have been developed to address different TE aspects in DCNs. In this chapter, we review the basic concepts of both networking and optimization that are used throughout the thesis. The first section presents an overview of data centers and different variants of the spanning tree protocol. Next, different classes of TE problem in DCNs are introduced with some intuitive examples. Then, we present the definition of Constraint Satisfaction Problem, followed by a description of Local Search. Finally, we present some background on Multi-objective Optimization. The related work is also mentionned in each section.

2.1 Data Centers and Switching Protocol variants

Data centers are now a key part of the Internet. Their number and their size are growing quickly. Some reports [KSG^{+}09, BAM10, BAAZ10] indicate that there are data centers containing up to 10K servers and some speculate that data centers could contain 100K servers or more. Data centers are used for various purposes. Some data centers are mainly used to perform computation while others are mainly used to provide Internet services. Many data centers support various applications at the same time and each application runs on a set of virtual machines that are distributed on physical servers.

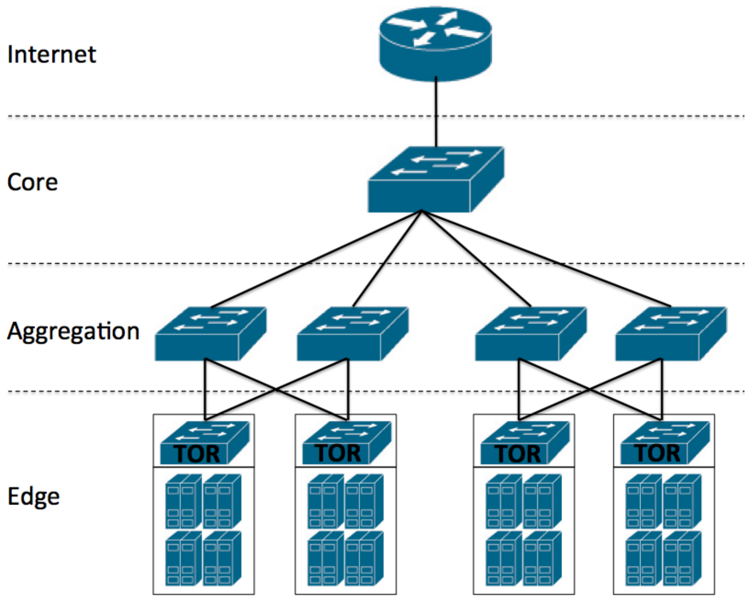


Figure 2.1: A data center network

From a networking viewpoint, data centers heavily rely on switched Ethernet networks. A typical 3-Tier Cisco topology [Sys10] for DCNs is illustrated in Figure 2.1. Servers are attached by using one or more Gigabit Ethernet interfaces to top of the rack (ToR) switches at the access (Edge) tier. ToR switches are connected to aggregation switches by using one or more 10 Gbps Ethernet links. These aggregation switches are then connected by using one or more 10 Gbps Ethernet links to core switches that are attached to routers when Internet access is required.

The switched Ethernet networks used in data centers are redundant to enable recovery in case of link or switch failures. However, currently deployed Ethernet switches can not directly use a mesh of links and need to use variants of the Spanning Tree Protocol [Soc98]. Several variants of the Spanning Tree Protocol are used.

2.1.1 Spanning Tree Protocol 802.1d

In Ethernet networks, multiple active paths between switches ensure the service availability in case of link failures. Unfortunately, Ethernet networks can not contain active cycles. To avoid such cycles, Ethernet switches implement the IEEE 802.1d Spanning Tree Protocol (STP) [Soc98] that reduces the topology of the switched network to a spanning tree. To compute the spanning tree, special messages called configuration bridge protocol data units (configuration BPDUs) are sent by the switches. Each of these messages contains the following information [Per99]:

- Root ID: ID of the switch assumed to be the root.
- Transmitting bridge ID: ID of the switch transmitting this configuration message.
- Cost: Cost of the least-cost path to the root from the transmitting switch.

The spanning tree protocol performs a distributed computation of the spanning tree. At first, each switch assumes itself to be the root, sets the Root ID to its own ID, sets the Cost value to 0 and transmits its configuration BPDU on every port. Configuration BPDUs are distributed on each LAN so that each switch receives the configuration BPDUs of its neighboring switches. The switch with the smallest ID is elected as the root. Thus, each time a switch receives such a message, it compares the Root ID contained in the message with its current Root ID. If the new Root ID is smaller, it generates its own configuration BPDU (Root ID, Transmitting bridge ID, and new Cost to the root) and transmits this BPDU on its ports. If there is more than one least-cost path to the root, the switch selects the one which passes via the lowest ID neighbor. The spanning tree is created when all the switches have selected the same Root ID and their cost of the shortest path to root are correctly calculated.

Table 2.1: Default link costs in 802.1d standard (M: Mbit/s, G: Gbit/s)

Data rate	4M	10M	16M	100M	1G	2G	10G
STP Cost	250	100	62	19	4	3	2

Obviously, the most important parameters that determine the resulting spanning tree are the switch IDs and the link costs. Network operators normally configure the switch with the highest capacity (ports $\times$ bandwidth) as the root of their spanning trees. By default, STP assigns a cost for each link based on its data rate (bandwidth). The link cost is an integer number in $[1..2^{16} - 1]$. Table 2.1 shows the 802.1d default cost of an interface for a given data rate. When the switch IDs and link costs are fixed, STP generates a specific spanning tree independently from the traffic demand matrices. Fortunately, one can configure the switch IDs and link costs to guide the STP standard to generate an intended spanning tree.

2.1.2 Multiple Spanning Tree Protocol 802.1s

The IEEE 802.1w Rapid Spanning Tree Protocol (RSTP) [Soc01] extends from 802.1d to accelerate the convergence of the alternate spanning tree in case of link failures or network topology changes. With 802.1d and 802.1w, the drawback is that only a small number of existed links is included in a single spanning tree since many available links are disabled.

The IEEE 802.1q Virtual LANs (VLANs) standard [Soc06] enables large Ethernet networks to be logically divided into many VLANs. VLANs are mainly used to isolate one application or one data center customer from the others. The servers (or virtual machines) that belong to a given VLAN can only communicate with the other servers that belong to the same VLAN. A switched Ethernet network can contain up to roughly 4000 different VLANs.

The IEEE 802.1s Multiple Spanning Tree Protocol (MSTP) [Soc02] is a combination of 802.1w and 802.1q enabling the service providers to benefit from the available links by spreading different spanning trees (one spanning tree per VLAN) over a single physical topology. Besides

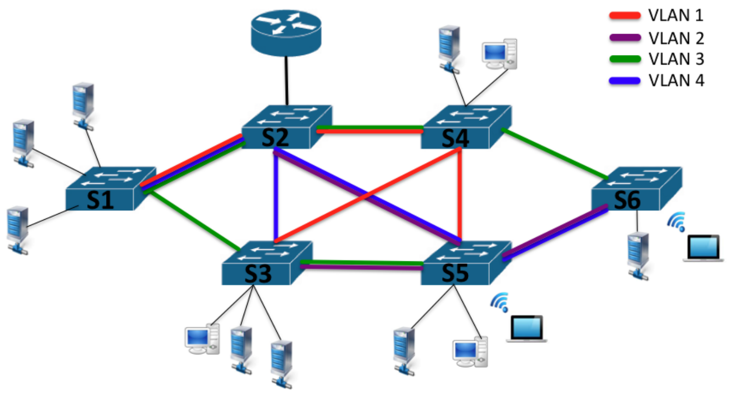


Figure 2.2: Example of Ethernet network with multi-VLAN

the big advantages enabled by the combination of two extensions of STP 802.1d, a huge configuration task is required in the network implementing MSTP. In practice however, it should be noted that most implementations of MSTP can only compute up to 16 different spanning trees, and map each VLAN onto one of these spanning trees. When many VLANs are defined, a common spanning tree is often generated for each set of VLANs in order to reduce the number of control frames in the network. The complexity of the MSTP increases when each VLAN requires a configuration of root and link costs to have its own spanning tree.

Example. Figure 2.2 depicts an Ethernet network implementing MSTP with four VLANs. Each VLAN is supposed to provide some specific customer services by grouping a set of connected switches in the network. In Figure 2.2, the spanning tree for each VLAN is represented with a specific color. Links are thus used by different VLANs in the network.

2.2 Traffic Engineering in Data Center Networks

There exist many criteria to classify TE problems for DCNs. If we simply rely on the traffic demand information, TE problems can be classified into two main categories as follows:

- **Off-line TE:** Demand matrices are given by doing an extensive study on the DCN traffic. Here, an off-line TE method is used to improve the performance of the switching protocols on the given traffic demand matrices. TE approaches for this category provide global solutions for the network design and configuration. The implementation of these approaches is often performed on cheap and normal switches without modifying the standard switching protocols.

- **Online TE:** The concept of online-TE was first presented by S. Kandula et al. in [KKDC05] in 2005 to dynamically adapt the routing when traffic changes in real-time. The network devices (router, switch) are required to be more intelligent and more powerful to perform the local adjustment due to the actual traffic situation of the network. Recently, the emergence of OpenFlow (flow-based) switches [MAB$^+$08] enables DCNs to support more flexible policies rather than standard switching protocols. However, this new generation network is currently in experimental phase and is very expensive for real world implementations.

In this thesis, we focus on the off-line TE category. The TE problem in large Ethernet networks such as DCNs is defined by an optimization of the network quality of service (QoS) based on many evaluation metrics such as link utilization, link delay, total network load, energy saving, number of used links, network convergence, service disruption in case of link failures, etc... Network operators can define their TE problem with one or a set of metrics to be optimized that depends on their actual needs. In practice, it is often impossible to optimize all these metrics at the same time because an improvement in one metric can degrade the value of other metrics. Single objective optimization can be performed to achieve the best solution for one metric while multiple-objective optimization aims to provide the best performance based on a compromise between a given set of metrics.

To classify TE approaches, different TE problem formulations divided the approaches into four main classes as follows:

- [Med06, HZC06, LYD$^+$03] aimed to map a set of VLANs to a given number of spanning trees.
- [dSS07, SdSA$^+$09, SdSA$^+$11] tried to assign a set of flows to a set of given spanning trees.
- [CJZ06] addressed the TE problem by building source-based multiple spanning trees (construct a spanning tree for each of the given source nodes).
- [PNM$^+$05, MSS09] advocated for solving the TE for MSTP by finding the set of spanning trees described by customer traffic demands and given network topology.

In this thesis, we follow the same approach as in [PNM$^+$05, MSS09]. In the remainder of this section, we introduce four TE metrics that we deal with in this research.

2.2.1 Maximal link utilization

The main objective in most TE problems is to avoid or minimize congestion. This is often evaluated by a minimization of the maximal or worst-case link utilization in the network. Many TE techniques in [PNM$^+$05, Med06, HZC06, dSS07, CJZ06, LYD$^+$03, SdSA$^+$09, MSS09, SdSA$^+$11] have been proposed for minimizing the maximal link utilization in the Ethernet networks with multiple spanning trees. The utilization of a link is defined as the ratio between its load and its bandwidth. With this definition, a link is overloaded if its load is greater than its bandwidth (link utilization > 1). The link load is computed as the sum of traffic flows on it. In DCN supporting many VLANs, the link load is aggregated from each VLAN using it. In Figure 2.2, the load of link (S1, S2) is aggregated from its load of VLAN 1, 3 and 4.

Example. The computation of the maximal link utilization is illustrated in Figure 2.3. To simplify the observation, we consider a network with one VLAN. We assume that each link in the network has the same bandwidth of 10 Gbits/s. By default, a link cost of 2 is thus assigned to every link. S2 is configured as the root because it has 4 links (like S3, S4, S5) and has the minimal switch ID. Thus, the spanning tree generated by STP with the default link costs (see Fig. 2.3a) is computed as follows:

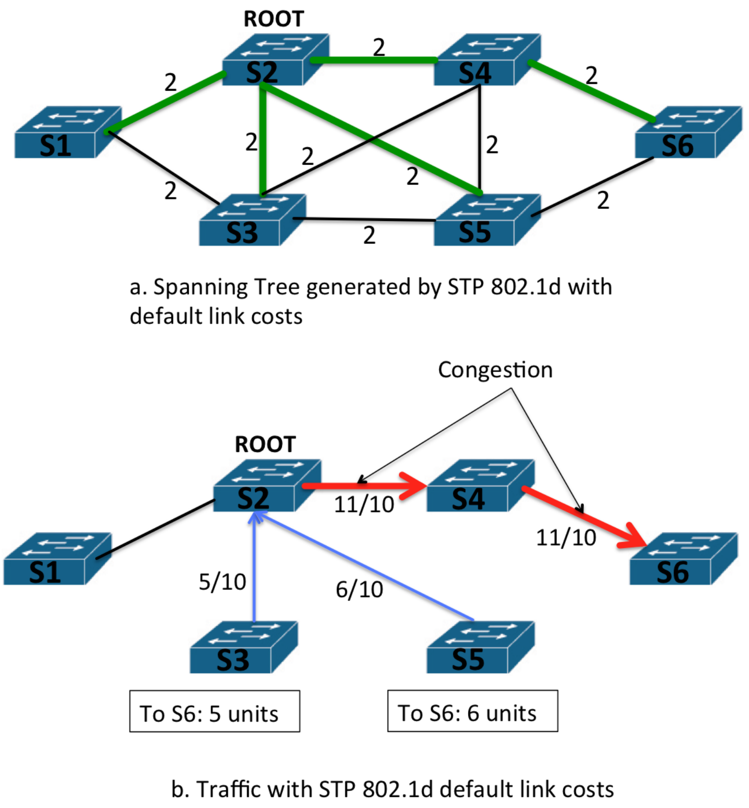

ROOT
S1
S2
S3
S4
S5
S6
2
a. Spanning Tree generated by STP 802.1d with default link costs
Congestion
ROOT
11/10
11/10
5/10
6/10
To S6: 5 units
To S6: 6 units
b. Traffic with STP 802.1d default link costs

Figure 2.3: Example of Ethernet network with default link costs

- Least cost path from S1 to root (S2): S1→S2
- Least cost path from S3 to root: S3→S2

- Least cost path from S4 to root: S4→S2
- Least cost path from S5 to root: S5→S2
- Least cost paths from S6 to root: S6→S4→S2 and S6→S5→S2. In this case, S6->S4->S2 is chosen because the ID of S4 is smaller than the one of S5.

The five links in the spanning tree are (S1,S2), (S2,S3), (S2,S4), (S2,S5) and (S4,S6). Assume that we have two demands: S3 wants to send 5 Gbits to S6 and S5 wants to send 6Gbits to S6. Let $U[i,j]$ denote the utilization of link (i,j). The utilization of each spanning tree link is depicted in Fig. 2.3b: $U[1,2] = 0$, $U[3,2] = 5/10$, $U[5,2] = 6/10$ and $U[2,4] = U[4,6] = 11/10$. Obviously, the maximal link utilization in this case is 11/10. This means that links (S2,S4) and (S4,S6) are overloaded.

A configuration of the link costs can guide the STP to avoid the congestion of the network in this case. Simply by changing the cost of the five links (S1-S3), (S2-S3), (S3-S5), (S4-S5) from 2 to 10 (see Fig. 2.4a), a new spanning tree is generated by STP. The maximal link utilization is reduced to 6/10. There is no congested links anymore in the network (see Fig. 2.4b).

2.2.2 Total load

The total load (or sum load) of a network is defined as the sum of all link loads in the network. Here, we reuse the example with the maximal link utilization to illustrate the computation of the total load in the network. Let $L[i,j]$ denote the load of link (i,j), $SumL$ denote the total load.

$$SumL = \sum L[i,j] \ (\forall switch i,j)$$

Example. In Fig. 2.3b, $SumL = L[3,2] + L[5,2] + L[2,4] + L[4,6] = 5 + 6 + 11 + 11 = 33$.
In Fig. 2.4b, $SumL = L[3,4] + L[4,6] + L[5,6] = 5 + 5 + 6 = 16$. In this case, the configuration of link costs can guide STP to reduce by more than 50% the total load.

The minimization of the total load is equivalent to the minimization of the average link load in the network. This TE problem with one

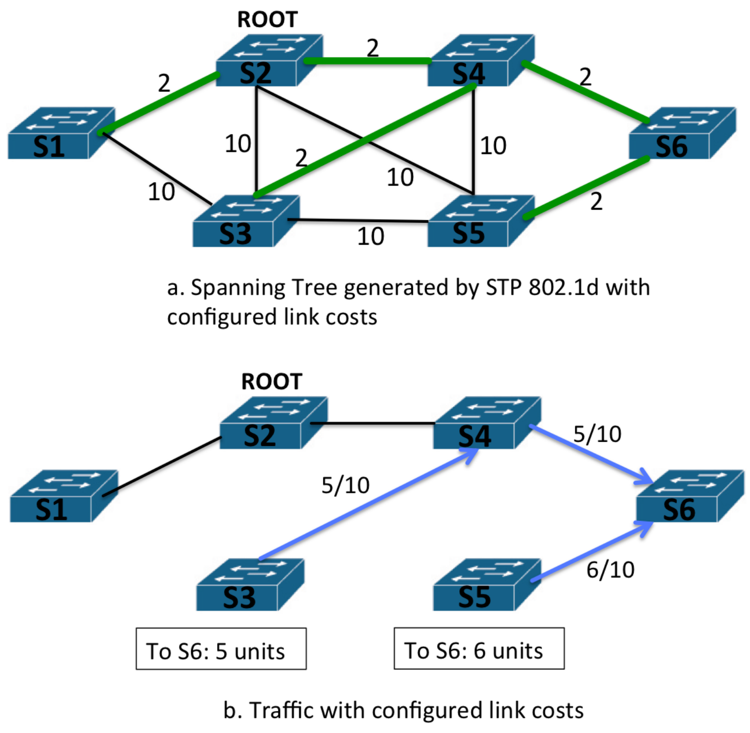

ROOT
S1
S2
S3
S4
S5
S6
2
2
2
2
2
10
10
10
10
10
a. Spanning Tree generated by STP 802.1d with configured link costs
ROOT
S1
S2
S3
S4
S5
S6
5/10
5/10
6/10
To S6: 5 units
To S6: 6 units
b. Traffic with configured link costs

Figure 2.4: Example of Ethernet network with configured link costs

spanning tree is known as the Optimum Communication Spanning Tree Problem (OCT) [Hu74] first proposed by T.C. Hu in 1974. OCT is a well-known NP-hard combinatorial optimization problem. Many exact methods have been proposed to solve this problem with small network instances of dozen nodes. In large DCNs supporting many VLANs, it is very challenging even to find a bound for this TE problem. Besides the maximal link utilization, total load is often an additional objective to evaluate the switching scheme performance.

2.2.3 Number of used links

Minimization of the number of used links is a new research topic in the network design domain. This metric is related to the energy efficiency (power consumption) of the network. Recent studies in [GMMO10, MSBR09a, MSBR09b, CSB$^+$08] aim to find a routing scheme that minimizes the number of active network elements including the number of used links in both IP and Ethernet networks.

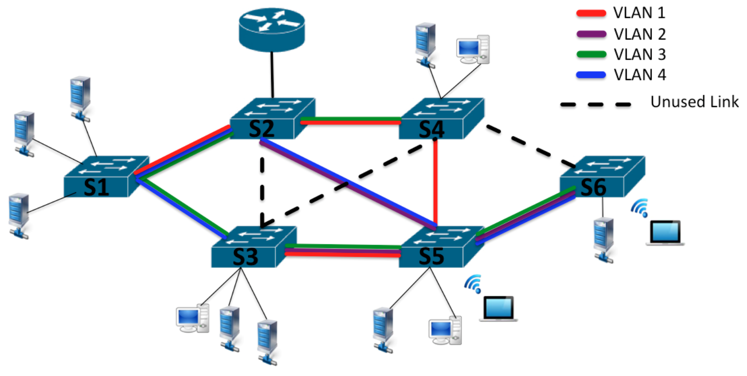


Figure 2.5: Minimization of number of used links

In DCNs using spanning trees, this metric is significant only if the number of VLANs is larger than one. However, to the best of our knowledge, there is no previous work dealing with the minimization of the number of used links in DCNs supporting many VLANs. In addition, the number of used links also has an impact on the fault tolerance (number of reserved links) and the network delay (the route lengths).

Example. In the Ethernet network of Fig. 2.2, all the links in the network are used by the spanning trees of its four VLANs. This solution is clearly not efficient in terms of power consumption, fault tolerance and path lengths. With the same network and the same VLANs, a better solution with four different spanning trees is illustrated in Fig. 2.5. We can see that three links (S3,S2), (S3,S4) and (S4,S6) can be disabled since they do not belong to any spanning trees.

2.2.4 Service disruption after link failures

The minimization of the service disruption is included in the TE problem presented by A. F. de Sousa and G. Soares [dSS07]. When MSTP detects a change of topology, i.e. the failure of some switches or some active links in the network, a recomputation of spanning tree is required in each VLAN affected by the failure. This minimization can be expressed as a minimization of the number of link replacements in each VLAN to maintain the spanning tree after failures.

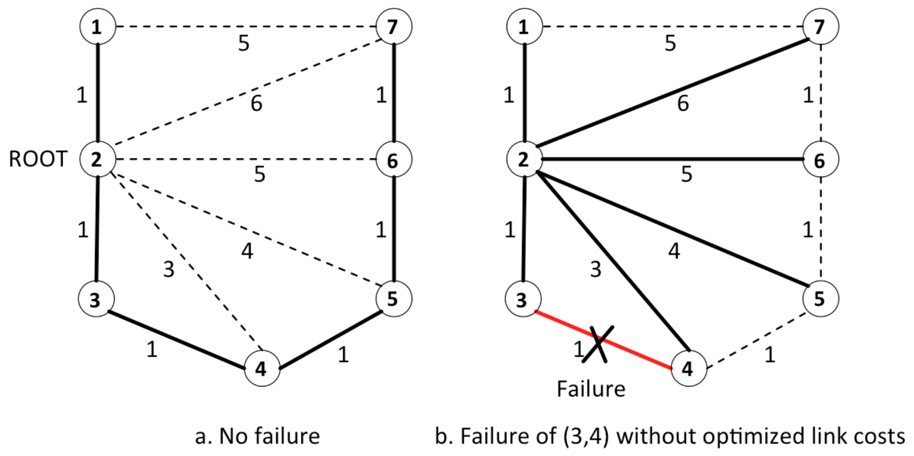


Figure 2.6: STP reconfiguration without a minimization of number of link replacements

Example. In Fig. 2.6, we illustrate the reconfiguration of the standard MSTP protocol after a failure of link $(3, 4)$ in a VLAN with 7 switches and an unoptimized link cost configuration. In Fig. 2.6a, each link of the spanning tree has a cost of 1, and the other links are assigned with a minimum cost such that MSTP will nerver include them into the spanning tree if there is no link failure. When the failure occurs to link $(3, 4)$ (see Fig. 2.6b), MSTP performs a recomputation of path cost from each switch to the root (switch 2). Then, only the links $(1, 2)$ and $(2, 3)$ are kept in the new spanning tree. All the traffic flows from/to the switches 4, 5, 6, and 7 are affected by this failure.

In Fig. 2.7, by performing a simple modification of the link costs, we can force MSTP to compute the same spanning tree if there is no link failure and one unique link replacement (link $(3, 4)$ is replaced by link $(2, 6)$) by MSTP after the failure of link $(3, 4)$.

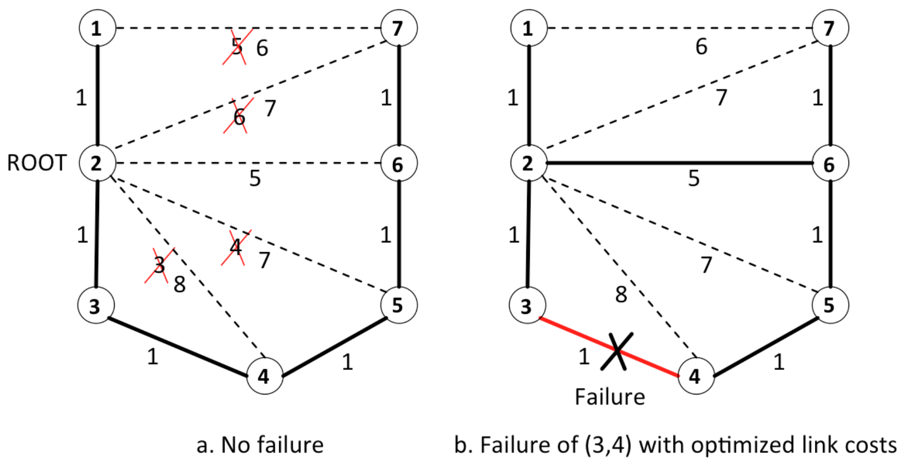


Figure 2.7: STP reconfiguration with optimized link costs

2.3 Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) $\langle X, D, C\rangle$ is defined by a set of variables $X = \{x_1, x_2, ..., x_n\}$, a set of domains $D = \{D_1, D_2, ..., D_n\}$ with D_i is the non-empty domain (possible values) of x_i, and a set of constraints $C = \{C_1(X_1), C_2(X_2), ..., C_n(X_n)\}$. Each constraint C_i involves a subset of variables $X_i \subseteq X$ and specifies the allowable combinations of values for the variables in X_i.

A solution to a CSP is an assignment of all variables such that all constraints are satisfied. Some CSPs also require a solution that maximizes or minimizes an objective function. Deciding the satisfiability of a CSP (i.e., the existence of a solution) is a NP-complete problem in the general case.

A Constraint Satisfaction Optimization Problem (CSOP) $\langle X, D, C, f\rangle$ is a CSP $\langle X, D, C\rangle$ with an objective function $f : X \rightarrow \mathbb{R}$ to be optimized. The objective is to find a solution to the $\langle X, D, C\rangle$ that optimizes (minimizes or maximizes) f.

In general, constraints in CSPs/CSOPs can be divided into hard constraints and soft constraints. Hard constraints must be satisfied by the solution while soft constraints are allowed to have a number of violations. Such an optimization with hard constraints is presented in Chapter 5.

2.4 Local Search

Local Search (LS) [MAK07] is an efficient method that can find high quality solution to CSP and CSOP in polynomial time. The search is called *local* because LS starts from an initial solution and it repeatedly performs changes on some features (local changes) of the solution to move to one of its neighbors. The search process ends when some criterion is met. In this section, we first present the concepts of neighborhood and move in LS. Then we describe some fundamental metaheuristics that can be combined with LS to improve the LS performance.

2.4.1 Principles

The neighborhood in LS is defined as a set of solutions that are produced from a given solution by modifying some of its features. A solution and its neighboring solutions normally share many common features. A move (or local move) in LS is a transformation from a solution into one of its neighbors. For instance, the neighbors of a spanning tree can be generated by doing a link replacement on it. In this case, its neighborhood is the set of spanning trees that differ from the given spanning tree by exactly two links.

LS algorithms often start with a random intitial solution or apply some heuristics to obtain a high quality intitial solution. The search process is done by repeatedly performing the following steps:

- **Neighborhood construction** is the key definition in a LS algorithm. It has an important impact on how the data structures and the solution space are organized and represented. In some cases, the definition of the solution and the neighborhood can reduce significantly the size of the search space and remove the symmetry of the problem. Neighborhood construction also influents the ease and the efficiency of the neighbor selection and the move strategy.

- **Neighbor selection** can be done randomly or based on a quality evaluation of each neighbor solution. For this step, heuristics such as first improvement, best improvement or random selection are used to choose the neighbor. In the problems with exponential search space such as TE in DCNs, the quality evaluation of the neighborhood is often very costly. A good design is required to guarantee the balance between the complexity and the efficiency. In this case, the incrementality is very important to speed up the computation of each objective value of the neighbor solutions before applying the move. Metaheuristics are often used in this step to guide the search. Several metaheuristics are described later in this section.

- **Move** transforms from the current soltuion into its selected neighbor. The computation task in this step consists of the local modifications of each solution feature and of the value of each objective function. An incremental adaptation is a good approach for computing only the local modifications of the data structures.

2.4.2 Metaheuristics

In LS, the search is performed *locally*. Thus, it is important to use the metaheuristics that guide the search to escape from the local optima or to explore the search space more efficiently. Hill-Climbing algorithm is a simple optimization technique in the family of LS. It starts with an arbitrary solution and iteratively performs the move to a neighbor with better solution quality. It stops when the *top of the hill* is reached - no futher improvement can be found. Unfortunately, this best found solution is often a local optimum.

Multi-Restarts

LS with Multi-Restarts is a useful technique that guides the search to escape from local optima. The LS algorithm is applied with different intial solutions to increase the diversification of the search. Different heuristics can be used for generating a good set of restart points. Fig. 2.8 depicts an example of LS with Multi-Restarts. With the different restart points, LS can explore different regions of the search space. Obviously, the opportunity to achieve the global optimum is larger.

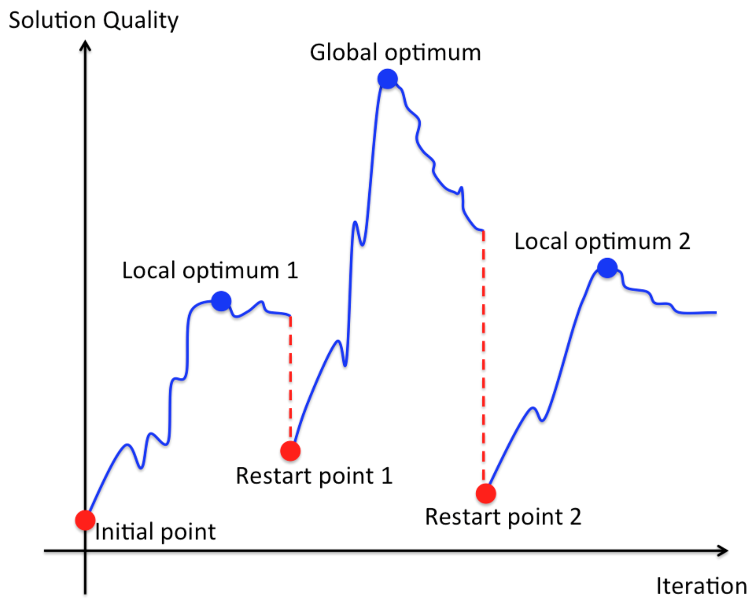


Figure 2.8: Local Search

Tabu Search

Tabu search [GL97] is a metaheuristic offering a diversified search in each of its search steps. The goal of tabu search is to prevent the search from visiting the same points in the search space. With tabu search, LS is often allowed to perform a move that degrades the quality of the current solution. This also prevents the LS algorithm from being stuck in the local optimum.

Simulated Annealing

Simulated Annealing (SA) [KGV83] is a probabilistic metaheuristic. The idea behind SA is to search for feasible solutions and to converge to an optimal solution. In each search iteration, it selects a random neighbor. If the solution quality of the neighbor is better than the current solution, then SA immediately performs the move. Otherwise, SA can accept a move with the degradation of solution quality based on some probability called acceptance criteria. The acceptance probability of a move with the degradation of solution quality is decreased during the search process until only improving moves are accepted. This is a very efficient technique to unstuck the search from the local optimum and to locate a good approximation to the global optimum. But the control of the acceptance criteria is often a difficult design.

Other metaheuristics are often used for solving the combinatorial optimization problems such as Genetic Algorithms, Variable Neighborhood Search or Ant Colony Optimization.

2.5 Multi-objective Optimization

In multiple-objective optimization [YST85], a solution that improves all the objectives should be preferred. But what if a solution's improving one of the objective values degrades the value of other objectives? For example, the number of used links and the sum load could be increased at the cost of reducing the maximal link utilization in the network. In the context of the TE problem, network operators prefer to provide the best (possible) QoS by striking a balance between the different objectives. Thus, it is necessary to define a different concept of "optimal" for multiple-objective optimization.

2.5.1 Pareto dominance and Pareto front

Given a minimization problem $\mathcal{P}$ with m objective functions, assume that there is no priority between these m objectives. Let $s_1 = (f_{1,1}, f_{2,1}, .., f_{m,1})$ and $s_2 = (f_{1,2}, f_{2,2}, .., f_{m,2})$ be the evaluations of two solutions for $\mathcal{P}$. We say that s_1 *dominates* s_2 (*Pareto dominance*) when s_1 is at least as good as s_2 for every objective and strictly better in at least one objective:

$$s_1 \succ s_2 \Leftrightarrow \forall i \in \{1, .., m\} : f_{i,1} \leq f_{i,2}, \exists k \in \{1, .., m\} : f_{k,1} < f_{k,2}.$$

Example. Fig. 2.9 depicts an example of a bi-objective solution space. By Definition 1, s_1 dominates s_7 but it does not dominate s_2, s_3, s_4, s_5,

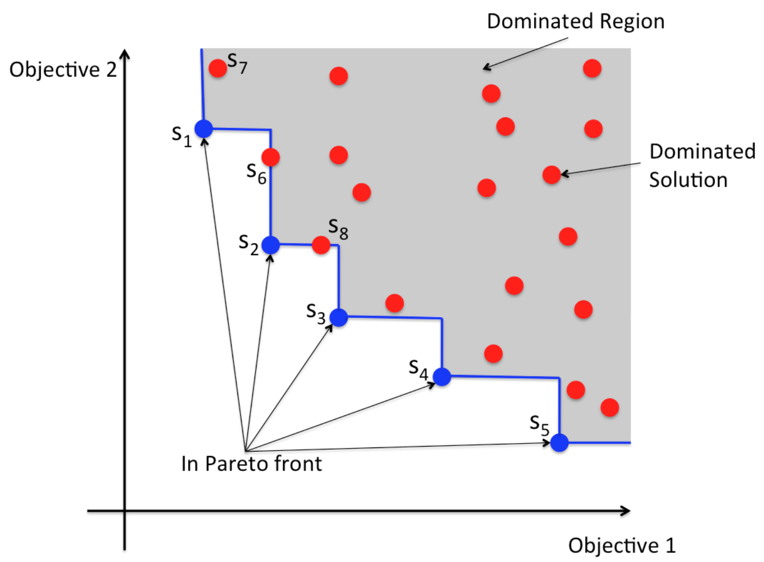


Figure 2.9: Pareto front with bi-objective optimization

s_6, and s_8. s_2 dominates both s_6 and s_8 even it has the same value of objective 1 with s_8 and of objective 2 with s_6.

Let $\mathcal{A}$ be a multi-objective algorithm for P and $\mathcal{F}$ denote the set of feasible solutions generated by $\mathcal{A}$ for $\mathcal{P}$.

Definition 2. *A solution $s^* \in \mathcal{F}$ is called Pareto optimal if s^* is not dominated by any solution in $\mathcal{F}$. The Pareto front of $\mathcal{A}$ for $\mathcal{P}$ is the set of all Pareto optimal solutions in $\mathcal{F}$.*

Hence, the output of multi-objective optimization is no longer a unique solution but a set of non-dominated solutions, i.e., the Pareto front.

Example. By Definition 2, the Pareto front PF in Fig. 2.9 contains five solutions s_1, s_2, s_3, s_4, and s_5 (the blue points). For each s_k containing in the dominated region (the red points) obtained with PF, we can find at least one solution s_i in PF such that s_i domniates s_k.

2.5.2 Multi-objective Optimization Methods

Different methods have been proposed for obtaining or approximating the Pareto front. The performance of each method depends on the kind of multi-objective combinatorial problem that it deals with and on the efficiency of its techniques.

- **Stochastic** [CCMn$^+$01, LBJT07] is an incomplet method (i.e. Local Search, Simulated Annealing, Tabu Search). Stochastic methods normally can not achieve the Pareto front but are very efficient for finding a non-dominated set that approaches the Pareto front for the problems with an exponential solution space.

- **Genetic Algorithm** [SD94, LCP11] is an incomplete, population-based method. Genetic algorithms start with an initial population (set of solutions) and iterately perform the selection and reproduction techniques to obtain a better population. This method is also efficient for finding a non-dominated set for the problems with an exponential solution space.

- **Linear Programming** [Zel74, lin99] is a fast, complete method but restricted to linearized problems and computationally difficult for large problems.

- **Preemptive Optimization** [CCK64, SS83] ranks its objectives by preference and optimizes them following this order. A good design is required to combine the algorithms for each single objective.

- **Constraint** [EZT00] selects one objective to be optimized, the other objectives are considered as the constraints to be respected. A bound should be defined in order to transform an objective into a constraint.

- **Weighted Sum** [KdW06] converts multiple objectives into a single objective by assigning an importance weight for each of its objectives and adding up all the objective functions.

- **Goal Programming** [NY06] aims to achieve a target value for each objective rather than optimize them. Each objective can be considered as a soft constraint.

Dealing with the multi-objective TE problem in DCNs, [HZC06] took into account link load balancing and admission fairness. The framework proposed in [LYD$^+$03] considers both network throughput and delay. Recently, [SdSA$^+$11] aimed to minimize the n worst link loads (with n up to the total number of network links) and the average link load in the network. However, the final output of these approaches is a unique solution (where the second metric is only considered as a constraint or a lexicographical objective) instead of a Pareto front. Thus, the assessment of these approaches is often performed by the separative observations on each of its objectives.

2.5.3 Local Search for Multi-objective Optimization

Stochastic methods, including Local Search [MAK07] are very efficient to approach the Pareto front by applying metaheuristics to efficiently explore the search space. Different approaches have been proposed for improving the performance of LS in dealing with multi-objective optimization.

- **Multi-objective Steepest Descent Method (MSDM)** [FS99] aims to find a *Pareto descent direction* for the search that maximizes the minimum degree of improvement of all objective functions. This can be done by solving a quadratic programming problem for finding the direction to move the current solution in each search step.

- **Evolution Strategies (ES)** [KC00] combine the Pareto archived evolution strategy LS with the use of population and recombination for solving a set of multiobjective knapsack problems. In each search step, ES randomly generates a new improving solution from the current solution according to a normal distribution.

- **Combined Objectives Repeated Line-search (CORL)** in [BdJ05] is a gradient-based LS method that computes the vectors for generating the convex cone of the *Pareto descent directions*. CORL is guaranteed to find the set of all non-dominated improving directions for any point in the parameter space of a multi-objective optimization problem.

- **Pareto Descent Method (PDM)** [HSK06] proposes a LS approach for moving solutions in the directions that simultaneously improve all objective functions. PDM finds the directions by solving linear programming problems.

The main drawback of these methods is that they are computationally expensive, restricted to some specific problems and inefficient for the multi-objective optimization problems with a huge solution space, such as TE in DCNs. In this thesis, we propose lightweight LS algorithms using efficient metaheuristics and speeding up techniques that can give good solutions for dealing with the TE problems in DCNs.

3 Ethernet networks using Spanning Tree

In this chapter, we first describe the TE problem in the Ethernet networks that use the spanning tree protocol. The objective is to minimize the maximal link utilization in the network. We then present how this network problem is modeled as a graph problem and how it is solved with our LS algorithm. Next, methods for generating different network topologies and traffic demand matrices are also developed. We conclude the chapter with an evaluation of the performance of our LS algorithm on synthetic data sets.

This chapter is based on our publications [HBD$^+$10b, HFDB09]

3.1 Introduction

During the last ten years, the Ethernet technology has replaced the other Local Area Network technologies in almost all enterprise and campus networks. Furthermore, many Service Providers deploy Ethernet switches in metropolitan and access networks. Last but not least, data centers are largely based on Ethernet switches. In many of these environments, the required bandwidth is growing quickly and network operators need to find solutions to ensure that their switched network can sustain the traffic demand without having overloaded links.

In this chapter, we deal with the TE problem in the Ethernet networks that implement the IEEE 802.1d Spanning Tree Protocol (STP) [Soc98]. STP reduces the set of links in the network into a single spanning tree topology to prevent the duplicate frames to be forwarded by the multiple active paths between switches. The creation of this spanning tree is based on the least cost (shortest) path from each switch to an elected root switch and is independent from the traffic demands in the network (see the description of STP in Section 2.1.1 in Chapter 2). However, STP standard itself does not provide any technique to configure the link costs for a given traffic demand matrix. Then our objective is to provide a TE technique that optimizes the link cost configuration in order to guide the STP to select a good spanning tree for a given traffic demand matrix.

Local search is a powerful technique that can find high quality solutions for computationally complex optimization problems (such as this NP-hard problem of traffic engineering) in polynomial time. Hence, we propose in this chapter a LS approach for solving this TE problem.

Searching on Link Costs or on Spanning trees?

To optimize the choice of the spanning tree by the STP protocol, two main approaches are possible.

- **Link cost optimization** is perfomed by searching the link cost space: try to change the cost of each link and thus the shortest path from each switch to the root switch. The cost of a link is an integer number in $[1..2^{16}-1]$. Even if we do not consider the choices of the root, the size of the search space in this case is of $(2^{16}-1)^m$ solutions (with m the number of links). In addition, it is difficult to control the impact of the change of the link costs on the spanning tree.

- **Spanning tree optimization** is done by directly searching on the spanning tree space. With this method, the size of the search space is reduced significantly but still exponential (i.e. $\binom{m}{n-1}$ with n the number of switches). Once the suitable spanning tree has been found, one has to determine link costs such that the STP yields the same spanning tree.

Solving this TE problem is a very difficult task as the search space is exponential. Exact methods are not appropriate especially for large instances. In this work, we focus on approximate methods based on LS with the spanning tree optimization approach.

Related Work

Many TE techniques have been proposed for IP, MPLS and optical networks in the last decade [NP09]. In IP networks, TE is usually performed by tuning the OSPF link weights [SGD05, For00]. In MPLS networks, TE techniques use constrained MPLS Labeled Switched Paths to redirect traffic flows around congested links [AmW99]. In optical networks, TE techniques allocate traffic flows to wavelengths [RS95]. Other TE techniques in [PNM+05, Med06, HZC06, dSS07, CJZ06, LYD+03, SdSA+09, MSS09, SdSA+11] have been proposed for solving the TE problem in Ethernet networks implementing the IEEE 802. 1s Multiple Spanning Tree Protocol [Soc02]. But to the best of our knowledge, there is no previous work dealing with the optimization on the link cost configurations for Ethernet networks using 802. 1d STP.

We evaluate our work by measuring the improvement of the solution quality in each test compared to the solution given by the STP standard. We also compare our solutions with the ones obtained with the IGP Weight Optimization (IGPWO) in [HFDB09]. The goal of this comparison is to see whether there is a large distance between our solutions and the ones of IGPWO where the routing is offering of Equal Costs Multi Paths with the participation of all links in the network.

3.2 Problem Formulation

We consider our Ethernet network as an undirected graph $G = (N, E)$ where N is the set of switches and E is the set of links between switches. Each link (s, t) has a bandwidth $BW[s, t]$ (we assume $BW[s, t] = BW[t, s]$). When link bundles are used between switches, we consider each bundle as a single link having the bandwidth of the bundle.

We call W the matrix of the link costs (see Table 2.1 in Chapter 2) for the default STP costs. $W[s, t]$ is computed from $BW[s, t]$. Let TD

be the matrix of the traffic demands. $TD[s,t]$ represents the traffic that switch s sends to switch t. Assume that a unique root is given, we call $STP(G,W)$ the spanning tree obtained by the Spanning Tree Protocol on graph G, with link costs W.

The traditional Ethernet switching problem can be defined as follows: for all pairs of nodes (s,t) so that $TD[s,t] > 0$, distribute the traffic demand over the unique path from s to t in $STP(G,W)$.

We call L the matrix of loads on each link (s,t) in the spanning tree: $L[s,t] = \sum$ flow over (s,t). For the computation of $L[s,t]$, the traffic flow is directed. This means that on link (s,t), $L[s,t]$ is different from $L[t,s]$.

The utilization of a link (s,t) is the ratio between its load and its bandwidth $U[s,t] = \frac{L[s,t]}{BW[s,t]}$. Like the computation of link load, the link utilization is directed. The link load is kept within the capacity if U[s,t]<1 and U[t,s]<1. If one of these two values goes above 100%, the link (s,t) is overloaded. These definitions of link load and utilization are equivalent to those used in [For00].

Our objective is to find a good configuration of link costs $W*$ minimizing the maximal utilization U_{max}:

$$U_{max} = max\{U[s,t] | s,t \in N\}$$

The formulation of this TE problem is the following:

PROBLEM 1
Input: Graph $G = (N,E)$, bandwidth matrix BW, traffic demand matrix TD
Output:

- A spanning tree SP^* minimizing U_{max}
- A link cost matrix W^* such that $STP(G,W^*)$ generates the spanning tree SP^*

3.3 Spanning Tree Optimization using Local Search

In this section, we first present an overview of our local search algorithm called LSA4STP (Local Search Algorithm for the Spanning Tree Protocol problem). Next we describe how to model and speed up queries on the spanning tree with a dynamic data structure. We then present the algorithm and the techniques allowing to break the symmetries, to define the neighborhood structure and to guide the search. Furthermore, we show how to incrementally compute the link loads with an efficient technique.

3.3.1 Algorithm Description

Algorithm 1 provides the Pseudo-code of our local search algorithm LSA4STP. Our LS algorithm aims to find the best (possible) solution in the spanning tree space. At each iteration, an edge replacement is performed in order to reduce U_{max}. The steps of LSA4STP, according to the Pseudo-code, are:

- Line 1: The method $getRoot(G)$ computes the $root$ switch - one of the two elements for creating spanning tree with the STP standard. We assume that this $root$ is unchanged during the search process. The detailed root selection is described in Section 3.3.3.
- Line 2: The method $getDefaultCosts(BW)$ returns the default link cost matrix W computed from the bandwidth matrix BW.
- Line 3: The initial spanning tree SP is computed according to the STP standard with the $root$ chosen in Line 1 and the link cost matrix W computed in Line 2.
- Lines 4 and 5: We store the initial solution obtained with the STP standard as the best solution at the start of the search. The method $getMaxUtilization(SP)$ returns the maximal link utilization U_{max} after computing the utilization of each link in SP.
- Line 6: We use time as the termination criterion. The search is iterated until a given time limit is reached.

Algorithm 1: Pseudo-code for LSA4STP

```
1  $root = getRoot(G)$;
2  $W = getDefaultCosts(BW)$;
3  $SP = STP(G, W)$;
4  $U^*_{max} = getMaxUtilization(SP)$;
5  $SP^* = SP$;
6  while $time\_exec < time\_windows$ do
7      $(s_O, t_O) = selectRemovingEdge(SP)$;
8      $(s_I, t_I) = selectAddingEdge(SP)$;
9      $SP = replaceEdge(SP, s_O, t_O, s_I, t_I)$;
10     $U_{max} = getMaxUtilization(SP)$;
11     if $U_{max} < U^*_{max}$ then
12         $U^*_{max} = U_{max}$;
13         $SP^* = SP$;
14     end
15 end
```

- Lines 7 and 8: With LSA4STP, the neighbors of a spanning tree can be generated by performing an edge replacement on it. Thus, an edge (s_O, t_O) (computed by the method $selectRemovingEdge$ in Line 7) will be chosen to be replaced by another edge (s_I, t_I) (obtained with the method $selectAddingEdge(SP)$ in Line 8). We describe the heuristics for selecting the edge to be removed and the edge to be added in Section 3.3.4.

- Line 9: An incremetal link load update is performed in the method $replaceEdge(SP, s_O, t_O, s_I, t_I)$ when (s_O, t_O) is replaced by (s_I, t_I). This incremental technique is presented in Section 3.3.5.

- Line 10-14: If the new U_{max} is less than the best known U^*_{max}, we store this solution SP as the best one SP^*.

3.3.2 Spanning Tree Modeling and Query

In our algorithm, we manipulate a spanning tree with two kinds of actions at each step of LSA4STP: (1) update the tree (i.e., edge replacement) and (2) query the tree (i.e., nearest common ancestor of two vertices, the father of a given vertex, etc...). Queries are usually performed many times in the neighborhood exploration phase. For instance, in order to determine whether or not an edge can be used to reconnect two disconnected sub-trees (by removing an edge of the current tree), we must check whether or not a given vertex belongs to a given subtree. This can be done by querying the nearest common ancestor of two vertices.

To simplify the design of our algorithm and to increase its efficiency, we use the VarSpanningTree abstraction of the $LS(Graph\& Tree)$ framework [DDVH09] for representing a tree of a given graph. $LS(Graph\&Tree)$ is a local search framework (an extension of COMET [VM05]) which simplifies the modeling of Constraint Satisfaction Optimization Problems on graphs and trees. Using this framework, many complex computations on trees are modeled and abstracted as a simple query. Finally, by using the incremental data structures (auto-update after each change of the tree), all queries on the spanning tree mentioned above can be performed in $O(1)$ time and the update action can be performed in $O(n)$ where n is the number of vertices of the given network.

3.3.3 Root Selection

Symmetry breaking [GS00] is a well-known technique in Constraint Satisfaction Problems to speed up the search. To determine a spanning tree, the STP needs an elected root switch and a link cost matrix. For a spanning tree SP containing $n-1$ edges, if the root is not fixed, there are n instances of SP: SP_1, SP_2,.., SP_n with n different roots (n the number of nodes). This means that our search space will be expanded to $n.\binom{m}{n-1}$ spanning trees (with m the number of links) if we must perform the search for all the different possible roots.

To eliminate all the symmetries in this problem, we fix a unique root during the whole search process. This has no impact on the single path between any pair of nodes in the spanning tree and has no impact on U_{max}. Although the root determination does not change solution, it can change the choice of the neighborhood solution in each search step. In addition, the root also influents the balance of the tree. Network operators normally configure the switches with the highest capacity (ports $\times$

bandwidth) as the root of their spanning trees. In our algorithm, we a priori select the switch having the maximal sum of associated link capacities (bandwidth) as the root. The method $getRoot(G)$ in Line 1 of Algorithm 1 returns such a root.

3.3.4 Neighborhood Formulation

In the search process, we try to move in the spanning tree space to find the solution with the smallest U_{max}. In this LS, two spanning trees are considered as neighbors if they differ by exactly two edges and they can be transformed into each other by performing one edge replacement (see Fig. 3.1). The principle of our algorithm is that in each search step,

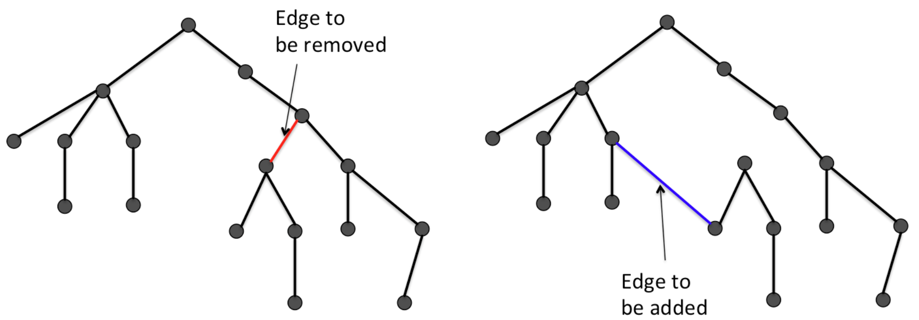


Figure 3.1: Edge replacement

we select an edge to be removed (Line 7 - Algorithm 1) and create a new spanning tree by adding another edge (Line 8 - Algorithm 1). In LS, the definition of the neighborhood is always a key design decision. Different heuristics can be used for selecting the edge to be removed and the edge to be added. However, the main challenge when dealing with this TE problem for large networks is to find good solutions in reasonable time. Thus, we must consider both the efficiency and the complexity of each heuristic to be applied in order to ensure the overall performance of LSA4STP. We present in this section the heuristics that give us the best experimental results.

Removing an edge

In order to reduce the maximal link utilization U_{max}, a natural question is: how to lighten the load of the most congested link in the spanning tree. An extreme solution is to replace this link with another one. Fig. 3.2 depicts our heuristic for electing an edge to be removed. To perform this task, LSA4STP accomplishes the following steps:

- First, we find the most congested oriented link (s_{max}, t_{max}) (U_{max} = $U[s_{max}, t_{max}]$) of SP that is not in the Tabu list (to be described later in this section).
- Second, from (s_{max}, t_{max}), we obtain a set SR of candidate edges to be removed that contains (s_{max}, t_{max}) and all the edges belonging to the subtree dominated by s_{max}, as illustrated in Fig. 3.2. Because (s_{max}, t_{max}) is the most congested link, we can assume that this congestion is caused by the traffic coming from the subtree dominated by s_{max}.
- Third, we denote TR the tree with root t_{max} containing the edges in SR. If we go from the leaves to root t_{max}, the more we climb the more the traffic increases. Our heuristic in this step is to assign to each edge in SR a probability to be selected based on its level in TR, as illustrated in Fig. 3.2. The edges closer to the root have a higher probability to be removed. The sum of the probabilities associated to the edges at level i is $\frac{1}{2^i}$, except at the last level d, where it is $\frac{1}{2^{d-1}}$ to ensure that the probabilities sum to 1.
- Finally, an edge (s_O, t_O) is selected based on these probabilities.

The idea behind here is to have an effective choice strategy: balancing intensification (greedy search for a solution) and diversification (consider unexplored neighborhood).

Adding an edge

After having removed (s_O, t_O) from SP, we obtain two separate trees. We denote TI the isolated subtree (unconnected to the root - see Fig.

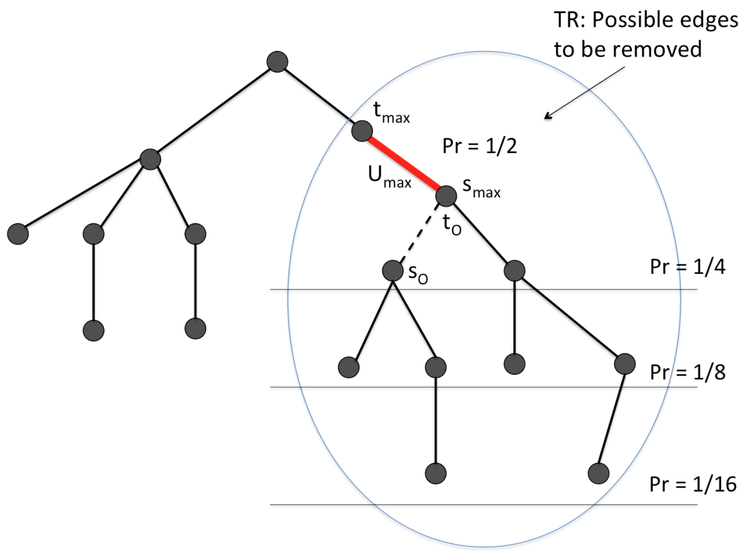


Figure 3.2: Selection of the edge to be removed

3.3). These two trees must be reconnected with an edge. Our objective is to have a solution with fewer congestion. For choosing an edge to be added to form the new spanning tree, LSA4STP accomplishes the following steps:

- First, we define a set of edges SA that contains all the edges that join $SP \setminus TR$ and TI.
- Second, we select k edges having the highest bandwidth and not in Tabu list from SA to form the set S_{maxBW}.
- Next, we compute the resulting U_{max} when adding each edge of S_{maxBW}. The speeding up techniques for this estimation of U_{max} are presented in Section 3.3.5.
- Last, the edge (s_I, t_I) in S_{maxBW} offering the minimal U_{max} is selected, as illustrated in Fig. 3.3.

For this step, we evaluate only k edges because it is more cost-effective than evaluating all the edges in SA. In our experiments, $(k = 10)$ gave good results.

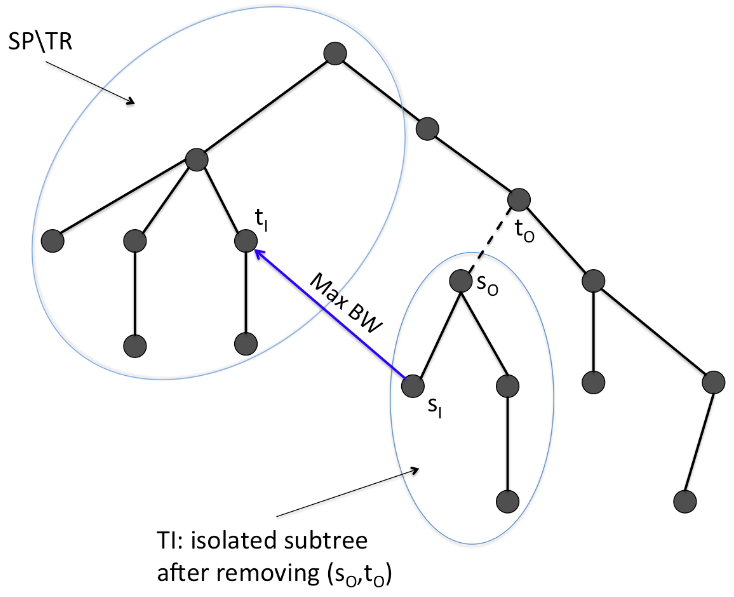


Figure 3.3: Selection of the edge to be removed

Tabu List

We use tabu search [GL97] - a heuristic offering a diversified search in each of its search steps. The goal of tabu search is to prevent the search from visiting the same points in the search space. In our problem, a solution is represented by a spanning tree. We can not store and mark all the visited spanning trees because of the expensive space complexity and time complexity to detect if a spanning tree has already been encountered. However, we implement tabu by forbidding the repetitive replacement of a couple of edges in successive iterations.

In LSA4STP, the considered (max congested) edge, the removed edge and the added edge are inserted into the tabu list at each search iteration. We freeze these edges for the next x search iterations. In our experiments, we obtained good results by setting $(x = 10)$.

3.3.5 Magic Cycle for Link Load Update

For the LS algorithm, it is important to visit as many points in the search space as possible. In this problem, the computation of link loads at each search step is a complex task. To compute the link loads, we must recompute all pairs paths in the spanning tree and then the loads over these paths. These computations have a costly time complexity of $O(n^2 log(n))$. Hence, an incremental link load update is very important to speed up the computation of U_{max} of a neighbor spanning tree before and after applying the edge replacement.

Algorithm 2: Pseudo-code for Link Load Recomputation

```
1 $L[s_O, t_O] = L[t_O, s_O] = 0$;
2 $cycle = computeCycle(SP, s_O, t_O, s_I, t_I)$;
3 foreach $(src, dest) \in TDSet$ do
4     if $pathInterCycle(SP, src, dest, cycle)$ then
5         $updateCycleLoads(src, dest, TD[src, dest], cycle)$;
6     end
7 end
```

However, we show that each time we replace an edge (s_O, t_O) by another edge (s_I, t_I), the load changes only on the links belonging to the cycle created by adding (s_I, t_I) (see the proof in Appendix A). Such a cycle is depicted in Fig. 3.4. This cycle is called 'magic' because it is very useful to speed up the LS algorithms proposed in this thesis.

By applying this incremental technique, we can avoid the all pairs paths recomputation at each search iteration. Algorithm 2 provides the Pseudo-code for computing the link loads with the following steps:

- Line 1: we assign 0 to the load of (s_O, t_O) because it is removed from the tree.
- Line 2: we compute the set of edges $cycle$ created by adding (s_I, t_I) to SP.
- Lines 3 and 4: we denote $TDSet$ the set of pairs of nodes $(src, dest)$ such that $TD[srt, dest] > 0$. For each pair of $(src, dest)$ in

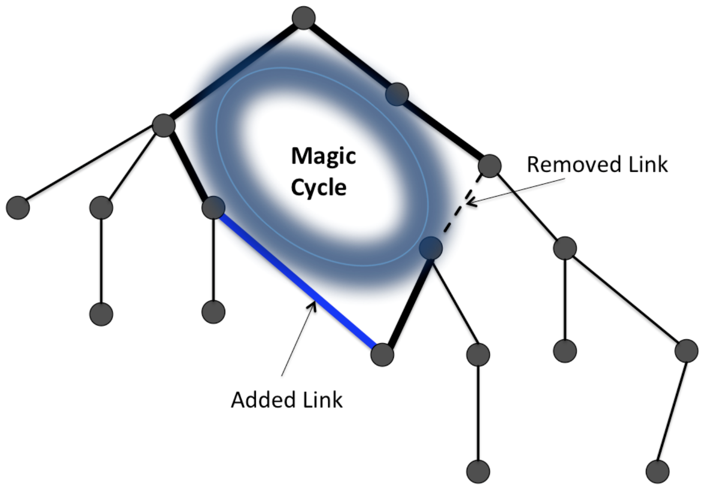


Figure 3.4: Cycle for updating link loads

$TDSet$, we verify whether the path from src to $dest$ includes the edges of $cycle$ or not .

- Line 5: the loads on $cycle$ is updated with each relating pair $(src, dest)$.

We can then benefit from the computations that we performed on the spanning tree before replacing (s_O, t_O) by (s_I, t_I) to compute the $cycle$ and its loads. The size (number of edges) of $cycle$ depends on the network topology. During our tests, we found $cycle$ that contained usually fewer than ten edges.

In our experiments, the technique speeds up the computation by about 20% of the execution time on each search iteration when the number of vertices is greater or equal to 100 nodes.

3.4 Link Cost Generation

As defined in Section 3.2, our objective is to find a good configuration of the link costs W^* such that $STP(G, W)$ yields a spanning tree minimizing U_{max}. In this section, we present a simple algorithm to generate the link cost matrix ensuring that the STP computes the intended spanning tree with low complexity.

Algorithm 3: Pseudo-code for Link Cost Generation

```
foreach Link $(s,t) \in E$ do
    if $(s,t) \in SP^*$ then
        $W^*[s,t] = W^*[t,s] = 1$;
    else
        $W^*[s,t] = W^*[t,s] = n$;
    end
end
```

From the spanning tree SP^* obtained by LSA4STP, we generate the cost matrix W^* by assigning a unit cost to all the edges in SP^* and by assigning a cost of n (number of nodes) to all the other edges in graph (see Algorithm 3). After this assignment, we can see that the cost of the longest possible path between a pair of nodes in any spanning tree is $n-1$ (passes $n-1$ edges) while the cost of the shortest path between any pair of nodes without using of spanning tree edges is at least n (passes one edge). Consequently, the 802.1d protocol will produce the intended spanning tree SP^*.

3.5 Data Generation

We present in this section the method for generating network topologies and traffic demand matrices for testing LSA4STP. We generated three types of generic topologies: Grid, Cube, Expanded Tree and use two data center topologies from the literrature: PortLand [NMPF$^+$09] and Fat Tree [AFLV08]. The traffic demand matrices were generated by using a uniform distribution for num_des (configurable) pairs of switches in the network.

3.5.1 Generic topologies

We present in this section three generic topologies used in our tests. The generation of these topologies is described hereafter.

Grid

In a grid topology, we consider each switch as a node on the grid. In Fig. 3.5a, we can see that in this grid topology, a node has at most four edges. The size of the square grid is x^2 with x being the number of nodes on a line. To generate a grid topology with n switches, we choose the smallest x such that $x^2 \geq n$. Among these x^2 nodes, we number the nodes increasing from left to right and from top to bottom. Our grid is the part of the square grid containing n nodes from node 1 to node n.

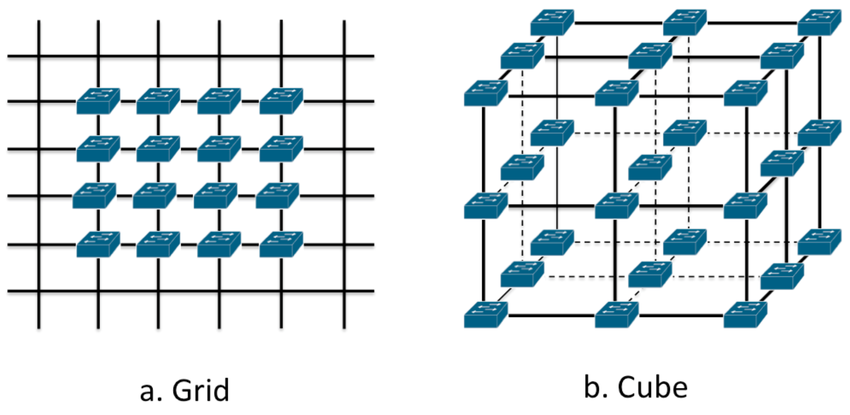


Figure 3.5: Grid and Cube topologies

Cube

We can consider a cube as a composition of x square grids, x being the number of nodes on a line (see Fig. 3.5b). As for the generation of the grid topology, we choose the smallest x such that $x^3 \geq n$. The cube network is the part containing the n first nodes from the cube.

Expanded tree

Tree or hierarchical network is a topology where there is no cycle and the nodes are organized by levels. The generation of a n-node tree topology is described in Algorithm 4. First, we fix the node 1 as the root of the tree (Line 1). At each next step, we randomly select a node u that has not yet been considered in the current tree (Line 5). The variable num_branch in Line 6 contains the number of child nodes of u. The value of num_branch is an arbitrary number in interval $[min..max]$. We set $(min = 2)$ and $(max = 6)$ to generate our tree topologies. Next, num_branch free nodes are selected randomly to be inserted into the tree by creating num_branch edges between u and these free nodes (Line 7 to Line 13). The node u is marked as considered so that we do not consider it any more in the next steps (Line 14). This step is iterated until all the nodes (from 1 to n) have been inserted into the tree (Line 4).

Algorithm 4: Pseudo-code for generating tree topology

```
1  $root = 1$;
2  $in\_tree = \{root\}$;
3  $considered = \emptyset$;
4  while $\#in\_tree < n$ do
5      select $(u \in in\_tree)$ and $(u \notin considered)$;
6      select $num\_branch \in [min..max]$ ;
7      foreach $i \in [1..num\_branch]$ do
8          if $\#in\_tree < n$ then
9              select $(v \in [1..n])$ and $(u \notin in\_tree)$;
10             $creatEdge(u, v)$;
11             $in\_tree = in\_tree + \{v\}$
12         end
13     end
14     $considered = considered + u$;
15 end
```

The tree obtained by Algorithm 4 is a spanning tree. We add to this tree two types of edges ensuring that we obtain a biconnected graph (see Figure 3.6). The advantage of a biconnected graph is: if any edge is removed, the graph remains connected. This property ensures that the STP can always be recomputed in case of link failures. In Figure 3.6, the edges of type (1) connect a leaf with a higher level node while the edges of type (2) connect a non-leaf node (except the root) with a same or lower level node of a different branch. For each tree, $(n-1)$ new edges are added to create the biconnected graph.

To simulate networks in which a switch has many ports, we define

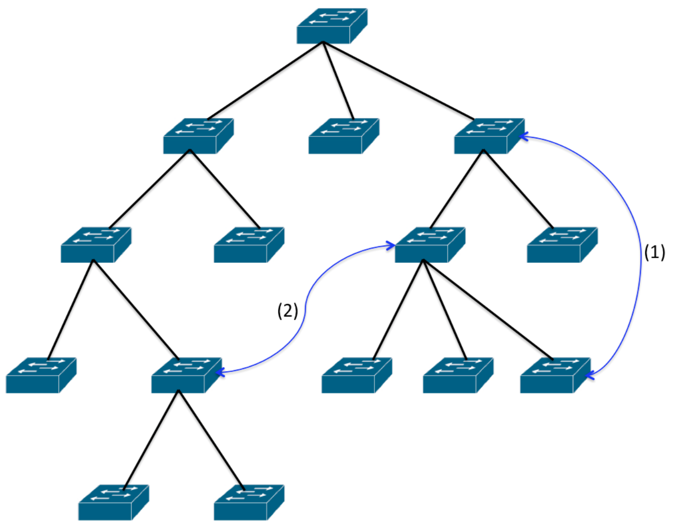


Figure 3.6: Expanded tree

a ratio r ensuring that each node in the tree is connected to at least r edges. In each test, from the generated biconnected graph, we create three more trees with ratio $r15 = n/15$, $r10 = n/10$ and $r5 = n/5$ (where n is the number of nodes).

Generating the link bandwidths and the link costs

For the topologies above, we use two volumes for the link bandwidth: Fast Ethernet 100 Mb/s and Gigabit Ethernet 1Gb/s. In our tests, 80% of the links are Fast Ethernet links and 20% of the links are Gigabit Ethernet links.

The initial link cost matrix W is generated based on the link bandwidths configuration BW. According to STP 802.1d, the default link cost for Fast Ethernet is 19 and 4 for Gigabit Ethernet (see Table 2.1 in Chapter

2).

3.5.2 Portland

Figure 3.7 depicts the PortLand DCN proposed in [NMPF$^+$09] consisting of 24 rows. There are 12 racks in each row. Each rack contains 40 servers interconnected by a ToR (top of rack) switch. Each ToR switch has 48 GigE ports: 44 GigE ports and 4 ports of 10 GigE. Each row has one EoR (end of row) switch containing 96 ports of 10 GigE. Each EoR switch connects to 12 ToR switches via one of the four ports of 10 GigE of the ToR. 24 EoR switches connect to a Core Switch Layer consisting of 1296 ports of 10 GigE. Thus we have 313 switches in a PortLand topology.

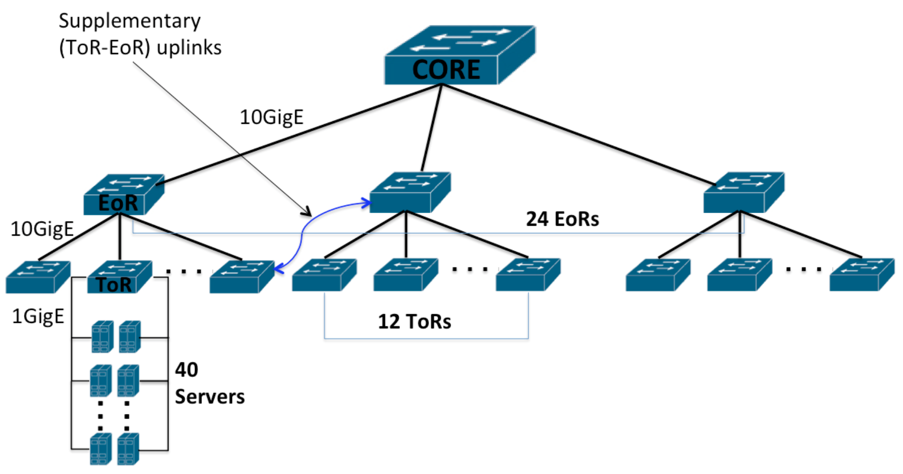


Figure 3.7: Portland

This PortLand design creates a 3-level spanning tree topology. As for the trees in the previous section, we do the same steps to obtain a biconnected graph. However, for the PortLand topology, we must consider the available ports on ToR, EoR and Core Switch Layer when adding edges. Because PortLand has only 3 levels, we aggregate the two types of edges (1) and (2) in Figure 3.6 as the blue edges between

ToR and EoR as depicted in Figure 3.7. Each ToR has 4 ports of 10 GigE, one of them is connected to an EoR, so we have 3 available ports of 10 GigE on each ToR. Each EoR has 96 ports of 10 GigE, 12 of these ports are connected to 12 ToRs. Another port is used to connect to the Core Switch Layer, so we have 83 free ports of 10 GigE for each EoR. Core Switch Layer has 1296 ports of 10 GigE, 24 ports are connected to 24 EoRs, and so it has 1272 free ports of 10 GigE. The generated biconnected graphs of PortLand are considered in our tests.

3.5.3 Fat Tree

Figure 3.8 depicts the Fat Tree - another topology for DCNs proposed in [AFLV08]. It is called Fat Tree because it is not a spanning tree like PortLand. All the ports on each switch are used to connect to the hosts or to the other switches. The Fat Tree topology is constructed as follows.

All the switches in Fat Tree have k ports. There are k pods. Each pod contains k switches divided into 2 layers: Aggregation and Edge layer. Each layer consists of $k/2$ switches. Each switch in the Edge layer connects to $k/2$ hosts and its remaining $k/2$ ports connect to $k/2$ switches of the Aggregation layer in the same pod. There are $(k/2)^2$ switches in the Core layer of a Fat Tree. Each Core switch connects to one switch in the Aggregation layer of all the k pods. We number the switches in the Aggregation layer of each pod from 1 to $k/2$. We split $(k/2)^2$ switches in the Core layer into $k/2$ portions, each containing $k/2$ switches. We number the switches in each of these portions from 1 to $k/2$. The i^{th} switch in each portion of the Core layer is connected to the i^{th} switch in the Aggregation layer of all the k pods. In our tests, we set $k = 16$, so we have 320 switches for each Fat Tree topology.

Link bandwidths and link costs generation

Like PortLand topologies, all the links of Fat Tree in our tests are 10 GigE and according to STP 802.1d, their link costs are 2.

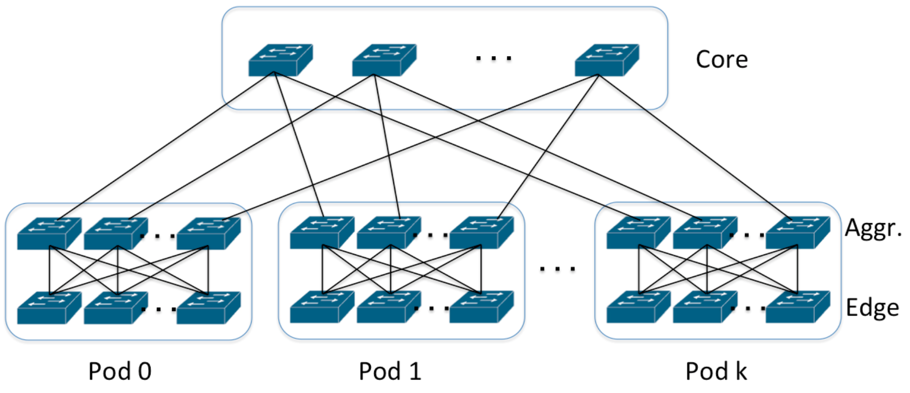


Figure 3.8: Fat tree

3.5.4 Traffic Demand Matrix

Several authors have recently analyzed the traffic matrices that are found in real data centers [KSG$^+$09, BAM10, BAAZ10]. Unfortunately, these datasets are for DCNs supporting many VLANs and there are no previous methods that can be used to generate traffic demands that are representative of data centers. For this reason, we wrote a simple traffic matrix generator that allows to test several types of traffic matrices. We first start with a uniform traffic matrix where all switches send traffic to all other switches in the network. Then, we generate traffic matrices that are less and less uniform by considering that most of the traffic is sent to a subset of the switches. These non-uniform matrices could correspond to storage servers or routers that often sink a large fraction of the traffic in data centers.

The pseudo-code in Algorithm 5 is used to generate the traffic demand matrix for all our tests. We consider that there is a subset of the switches that receive most traffic. In practice, these switches would be the ones attached to routers in data centers or the ones attached to high end servers in Campus networks. The method $getDestinationsSet$ in Line 1 returns the set of destinations for which all switches send the traffic to. From the input parameter num_des, this method generates the set of destinations by selecting randomly num_des nodes out of n

Algorithm 5: Pseudo-code for generating Traffic Demand matrix

```
1 $destinations = getDestinationSet(num\_des)$;
2 foreach $i \in [1..n]$ and $j \in desinations$ do
3     if $i \neq j$ then
4         $TD[i,j] = getUniformTD(minTD, maxTD, sumTD)$;
5     end
6 end
```

Table 3.1: Data Generation and Time Window for LSA4STP

No	Topo. Type	Num. Nodes	sumTD(Mb)	Time Window (s)
1	Grid	50	400	300
2	Cube	50	400	300
3	Expanded Tree 1	50	1000	300
4	Expanded Tree 2	100	1500	600
5	Expanded Tree 3	200	2600	1200
6	PortLand	313	120000	1800
7	Fat Tree	320	56000	1800

(number of nodes) to insert into $destinations$.

In our experiments, we generate for each network topology five traffic demand matrices:

- All switches receive traffic - uniform matrix $(num\_des = n)$
- 50% of switches receive traffic $(num\_des = n/2)$
- 20% of switches receive traffic $(num\_des = n/5)$
- 10% of switches receive traffic $(num\_des = n/10)$
- 5% of switches receive traffic $(num\_des = n/20)$

For each network topology, we fix a sum of traffic demand ($sumTD$ in Table 3.1) for all of these five traffic demand matrices. This $sumTD$

depends on the network type, network size, number of links and link bandwidths (see Table 3.1). For each pair of switches, the method $getUniformTD$ in Line 4 generates a traffic demand in the interval $[minTD..maxTD]$. The $minTD$ and $maxTD$ values are computed based on $sumTD$.

3.6 Experiments and Results

We generated the seven topologies described in Table 3.1. Each topology has five traffic demand matrices as described in Section 3.5.4. For each Expanded Tree, we have four topologies (biconnected graph, $r15 = n/15$, $r10 = n/10$ and $r5 = n/5$) with the same traffic demand matrix. The input of each test is one network topology - Graph $G = (N, E)$, one bandwidth matrix BW, and traffic demand matrix TD. We generated 50 tests for Cube (10 topologies x 5 traffic demand matrices), 50 tests for Grid (10 topos x 5 tdms), 200 tests for each of Expanded Tree from 1 to 3 (40 topos (10 for each of biconnected graph, $r15 = n/15$, $r10 = n/10$ and $r5 = n/5$) x 5 tdms), 50 tests for Fat Tree (10 topos x 5 tdms), and 50 tests for PortLand (10 topos (biconnected graph) x 5 tdms). So we have 800 tests. These tests are available online in [HBD$^+$10a]. The time window for running LSA4STP for each topology type is described in Table 3.1. In this thesis, the execution of our experiments is performed on a high-throughput environment, made of 30 cores distributed on a set of Intel(R) Core(TM) 2 Quad CPU at 2.40GHz with 1GB RAM for each core. Each algorithm is allocated one core, and the distribution is managed by Condor(R) [1].

In this section, we analyze the obtained results by evaluating the improvement of the maximal link utilization in each test and by comparing this solution with the one obtained with IGPWO in [HFDB09] assuming that switches would be replaced by routers.

[1]http://www.cs.wisc.edu/condor/

3.6.1 Improvement compared to STP 802.1d default solution

We consider the improvement of the maximal utilization U_{max} as the criterion to evaluate the performance of LSA4STP. We also measure U_{max} obtained with the default costs by STP 802.1d.

We present in Fig. 3.9 our results obtained for the two generic networks: Cube, Grid as well as Fat Tree and PortLand. For these four networks, LSA4STP gives the best results for the Cube since it reduces on average the value of U_{max} up to 50% for all the five types of traffic demand matrices. For the Grid, Fat Tree and PortLand, U_{max} is also reduced to about 60%-80%. LSA4STP works better for the Cube, Grid

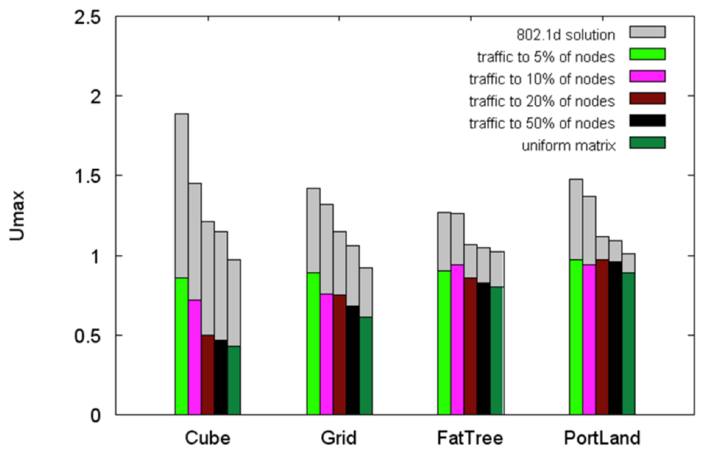


Figure 3.9: Result for Cube, Grid, Fat Tree & PortLand

than for the Fat Tree, PortLand because the link bandwidths in the Fat Tree, PortLand are homogeneous (all 10 Gb/s) while the ones in the Cube, Grid are not (100 Mbs and 1 Gb/s). But the reason is not the difference of the sum of link bandwidths between the spanning trees. In our tests, this sum of the initial (802.1d) and best U_{max} (LSA4STP) one is almost the same. However, for the Cube and Grid, in the 802.1d solutions the most congested links are the links of 100 Mb/s while the 1 Gb/s ones are not efficiently used as in the solutions obtained with LSA4STP.

Our local search algorithm is especially efficient for the tests of Expanded Tree in which high quality solutions exist in a large search space. Fig. 3.10 depicts the results of LSA4STP with the tests of Expanded Tree 100 in which U_{max} is dropped to about 49% for the Biconnected Graph, 47% for $r15$, 64% for $r10$ and 78% for $r5$. The reason is always the efficient use of 1Gb/s links. With the ratio $r5$, the average number of 1 Gb/s links is 97 out of 99 links of spanning tree (almost homogeneous of link bandwidths). This number of Biconnected Graph is 32, of $r15$ is 53, of $r10$ is 76. That explains why with the ratio $r5$, LSA4STP has the similar results as Fat Tree and PortLand.

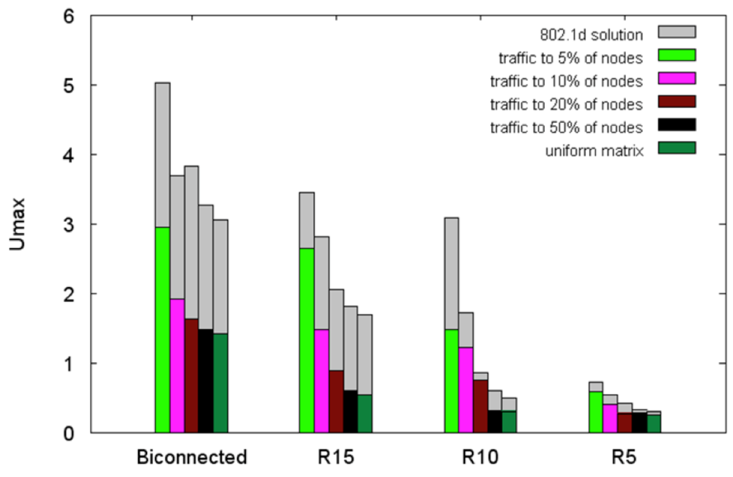


Figure 3.10: Result for Expanded Tree 100

In both of Fig. 3.9 and Fig. 3.10, we can see the same decreasing order of U_{max} for each test from left to right. With the same topology and the same sum of traffic demand ($sumTD$), each traffic demand matrix gives a different U_{max} (obtained with 802.1d or LSA4STP). Higher U_{max} values are observed with the more biased traffic demand matrices. Because for all these five traffic demand matrices, all switches send traffic to a number of switches (num_des). So the more switches receive traffic the more balancing traffic is distributed to the links. Obviously, the uniform traffic demand matrix always gives the lowest U_{max}.

We describe in Table 3.2 the average time for LSA4STP to find the

Table 3.2: Average time for LSA4STP finding the best solution (in s). ($n\ des \rightarrow \#des = n$)

Topo. Type	$n\ des$	$n/2\ des$	$n/5\ des$	$n/10\ des$	$n/20\ des$
Grid	192	188	170	185	124
Cube	91	165	130	130	206
Exp.Tree 1	108	103	130	128	143
Exp.Tree 2	222	224	211	306	271
Exp.Tree 3	251	232	298	321	258
PortLand	350	339	418	427	412
Fat Tree	409	392	438	385	395

best solution for each test of the seven topologies. This time for the topologies of: 50 nodes (Cube, Grid, Expanded Tree 1) is about 2 minutes, 100 nodes (Expanded Tree 2) is ∼4 minutes, 200 nodes (Expanded Tree 3) is ∼5 minutes and 313 (PortLand) and 320 (Fat Tree) nodes are ∼6 minutes. We can state that LSA4STP works well for the large scale tests when the time complexity of link load computations in each search iteration is collapsed.

3.6.2 Comparison with IGP Weight Optimization

Several IGP Weight Optimization (IGPWO) techniques have been proposed for IP networks [For00, HFDB09, SGD05, NP09]. In this section, we compare the performance of LSA4STP to a local search algorithm (LSA4IGPWO) in the COMET language [HFDB09]. IGP weight optimization is not applicable to existing Ethernet networks as it only applies to IP routers. Compared to Ethernet switches, IP routers have the advantage of being able to send packets over all links in the network while the STP disables a subset of the links. Given the price difference between IP routers and switches, large datacenters will not replace their switches with IP routers. However, there is ongoing work within the IETF to develop standards to allow next-generation switches to use the IS-IS routing protocol instead of the 802.1d protocol [Per04, Per10]. This solution requires more powerful switches and it can be expected that it will only be supported on new high-end switches initially.

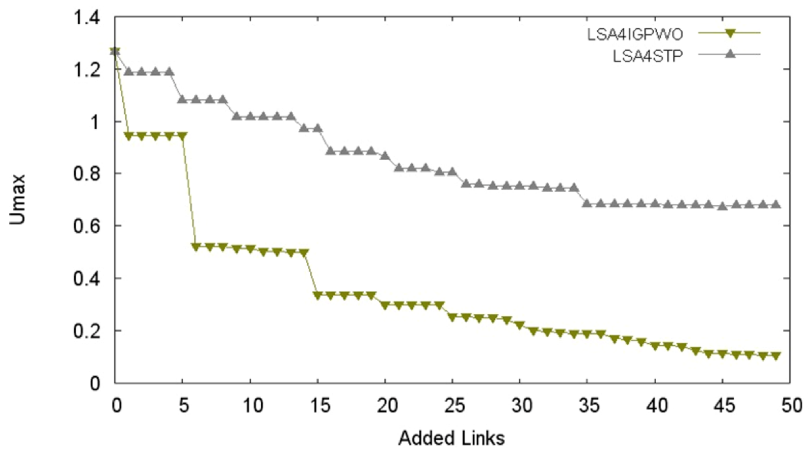


Figure 3.11: LSA4STP vs LSA4IGPWO

To compare IGPWO with LSA4STP, we performed an experiment with the Expanded Tree containing 50 nodes and the uniform traffic demand matrix ($num\_des = n$). The time windows for running IGPWO and LSA4STP are the same (300s). To evaluate the impact of the number of alternate paths in the topology, we varied the number of additional links in the Expanded Tree. Figure 3.11 shows on the x axis the number of links that were added to the tree (see Section 3.5.1) and on the y axis the maximal link utilization for IGPWO and LS4STP. When there are fewer than 6 links, LS4STP is within 20% of the solution obtained by IGPWO. When the number of additional links in the Expanded Tree grows, the distance between LSA4STP and IGPWO grows as well. This is normal since LSA4STP uses only a fraction of the links while IGPWO is able to send traffic over all links. For example, regarding Figure 7, at 25 added links, to obtain an U_{max} ~0.3, IGPWO must use all the 74 links. So when there are link failures, this solution given by IGPWO has no reserved link to ensure the service availability while the number of reserved links of the one obtained with LSA4STP is 25.

3.7 Conclusion

In this chapter, we dealt with the minimization of the maximal link utilization U_{max} in Ethernet networks containing one spanning tree. We proposed a new TE technique based on local search that finds the best (possible) spanning tree that minimizes congestion for a given traffic matrix. Our choice of directly optimizing spanning trees instead of link weights reduces considerably the size of the search space. We proposed an efficient technique to recompute the link loads called "magic cycle" at each search iteration that can avoid the all pairs paths computation. This cycle is very useful and it is used by all our algorithms in this thesis.

From the LS techniques viewpoint, this chapter can be considered as the heart of the thesis. The promising results obtained with LSA4STP motivate us to continue with local search to cope with other TE problems in large Ethernet networks. Our LS algorithm in next chapter deals with the minimization of U_{max} for DCNs containing multiple spanning trees.

4 JOINT OPTIMIZATION OF MULTIPLE SPANNING TREES

In this chapter, we deal with the minimization of the maximal link utilization in large DCNs containing multiple spanning trees. This TE problem in Ethernet networks is now more challenging in large DCNs with more switches and many spanning trees are required to be generated. We analyzed different studies on DCNs to obtain data sets that are representative of data centers. The description of our LS algorithm is presented, followed by a performance evaluation. We demonstrate the efficiency of our LS approach comparing to the Multiple Spanning Tree Protocol 802.1s standard [Soc02].

This chapter is based on our publication [HDBF11b].

4.1 Introduction

Current powerful data centers with thousands of servers have the proficient ability to provide computation, data access, storage and sharing services that allow the web-based and multi-media applications to expand quickly in the trends of cloud computing. Dealing with the provisioning and scalability problem, data centers are largely based on Ethernet switches with many Gigabit Ethernet interfaces. Spanning Tree Protocol (STP) 802.1d [Soc98] and its variants have heavily dominated in DCNs for a couple of decades. In Chapter 3, we studied the Ethernet and DCNs that implement STP to restrict the network to a single spanning tree rooted on, typically a core switch. The major drawback of the STP is that it disables the links that do not belong to the selected spanning tree. This is potentially a waste of resources since these links exist in the network. Fortunately, the IEEE 802.1s Multiple Spanning Tree Protocol (MSTP) standard [Soc02] enables the network operators to divide the whole DCN into different overlapping regions called Virtual Local Area Networks (VLANs) [Soc06]. With MSTP, a spanning tree can be created for each VLAN by computing the least-cost path from each switch to an elected root switch. The details of the MSTP have been described in Chapter 2. However, MSTP itself cannot ensure a good set of spanning trees for a given set of traffic demand matrices because it inherits the same configuration method of the spanning tree from STP.

Related Work

There exist four main approaches to deal with the TE problem for MSTP.

First, MSTP optimization techniques in [Med06, LYD$^+$03, HZC06] aimed to map a set of VLANs to a given number of spanning trees.[Med06] introduced two algorithms: a Multiple Spanning Tree Generation Algorithm (MSTGA) to generate a set of spanning trees that have the smallest number of links in common (or maximum edge-disjoint trees) and a VLAN-Spanning Tree Mapping Algorithm - a greedy algorithm to map each VLAN to a spanning tree generated by MSTGA and to minimize the number of used links. A tiny network with 7 nodes and 3 VLANs was considered to evaluate the perfomance of their algorithms. Yujin et al. propose in [LYD$^+$03] a QoS-aware mechanism that allows to tune the link costs and to provide VLAN mapping policy based on the traffic characteristics for minimizing network throughput and delay. The admission control algorithm in [HZC06] assigns a group of VLANs to each given spanning tree and then maps each service request to a VLAN in order to minimize link loads and to admit maximal amounts of customer services. This algorithm gave good results with the small networks of 5, 10, 15, 20, 25 nodes in full mesh.

Second, [dSS07, SdSA$^+$09, SdSA$^+$11] proposed heuristic schemes for ensuring the load balancing in Metro Ethernet using MSTP by mapping a set of traffic flows to a set of given spanning trees. Different criteria were taken into account to be optimized such as service disruption and worst-case maximal link utilization in [dSS07], a given number n of worst link loads and average link load in [SdSA$^+$09, SdSA$^+$11]. However, these techniques are only applicable for small networks (up to 23 nodes).

Third, the construction algorithm proposed in [CJZ06] addresses the TE problem for the US network with 12 vertices and 17 links by constructing a spanning tree for each of the given source nodes (the nodes with non-zero total traffic requests). The drawback of this approach is that the number of spanning trees to be constructed must be equal to the number of nodes that send traffic in the network.

Last, [PNM$^+$05, MSS09] advocated for solving the TE problem for MSTP by finding the set of spanning trees in the metro domain that provides load balancing for a given set of customer traffic demands. The heuristic proposed in [PNM$^+$05] tried to assign weights to the links in the networks. The limitation of this approach is that it is difficult to control the impact of the weight updates on the creation of each spanning tree. In [MSS09], the Best Multiple Spanning Tree algorithm aims to find the best set of Edge-disjoint spanning trees, and the best mapping of the VLANs to that set of disjoint spanning trees among all other possible cases in a given network. Thus, its complexity is too large.

The main drawback of the approaches mentioned above is that they are not applicable for large networks. In this work, we solve the same TE problem as in [PNM$^+$05] but we aim to minimize the maximal link utilization for DCNs. To our knowledge, our solution is the first technique that allows to release the choices of spanning trees from the dependence on link weights. Our scheme can tackle the TE problem for large DCNs containing hundreds of switches instead of small networks as in the state-of-the-art techniques.

Our LS algorithm LSA4STP proposed in Chapter 3 considered the TE problem for the STP 802.1d in switched Ethernet networks containing one spanning tree. The promising results obtained with LSA4STP motivate us to extend it for solving the same problem of minimization of U_{max} but coping with data centers containing many VLANs (spanning trees). In addition, the topologies and traffic demand matrices of current data centers mentioned in [BAM10] are used for evaluating the performance of our algorithm.

4.2 Problem Description

We consider our Ethernet network as an undirected graph $G = (N, E)$ where N is the set of switches and E is the set of links between switches. We call BW the matrix of link bandwidths. Each link $(i, j) \in E$ aggregates all link bundles between i and j into a single link having the total bandwidth of the bundles (note that $BW[i, j] = BW[j, i]$). Let $V = \{V_1, V_2, \ldots, V_k\}$ be the set of k given VLANs in the network, with $V_r = (N_r, E_r)(N_r \subseteq N$, $E_r = \{(i, j) \in E | i, j \in V_r\}$, $1 \leq r \leq k$). Each graph V_r is assumed to be connected. We call $TD = \{TD_1, TD_2, \ldots, TD_k\}$ the set of k traffic demand matrices where $TD_r[i, j]$ represents the traffic that switch i sends to switch j in VLAN V_r ($1 \leq r \leq k$; $i, j \in V_r$).

Let $W = \{W_1, W_2, \ldots, W_k\}$ be the set of link cost matrices for the k given VLANs. We call $MSTP(G, V, W)$ the set of k spanning trees $SP_1(G, V_1, W_1)$, $SP_2(G, V_2, W_2)$,..., $SP_k(G, V_k, W_k)$ obtained with the Multiple Spanning Tree Protocol on graph G, with set of VLANs V and set of link cost matrices W.

The Ethernet switching problem here is to distribute all the traffic demands $TD_r[i, j] > 0$ ($1 \leq r \leq k$; $i, j \in V_r$) over unique path from i to j in $SP_r(G, V_r, W_r)$. Obviously, the only set that can be configured to change the solution (set of k spanning trees) is W.

We reuse the definitions of link load L, link utilization U and of U_{max} as described in Section 3.2 in Chapter 3. Here, the values of $L[i, j]$ and $U[i, j]$ on each link (i, j) in E are computed from the aggregation of the traffic of all the VLANs that contain (i, j) in their spanning tree.

The formulation of this TE problem is the following:

PROBLEM 2
Input: Graph $G = (N, E)$, set of k VLANs V, bandwidth matrix BW, set of k traffic demand matrices TD

Output:

- A set of k spanning trees SP^* minimizing U_{max}
- A set of k link cost matrices W^* such that $MSTP(G, V, W^*)$ generates the k spanning trees SP^*.

The link cost generation problem is solved by applying the technique proposed in Section 3.4 in Chapter 3 for each VLAN in V.

4.3 MSTP Optimization Using Local Search

In this section, we present our local search algorithm called LSA4MSTP (Local Search Algorithm for the Multiple Spanning Tree Protocol problem). LSA4MSTP is extended from LSA4STP in Chapter 3. In LSA4MSTP, we perform the search directly on the spanning trees instead of link costs. This reduces significantly the size of the search space from $(2^{16} - 1)^{k.m}$ to $\binom{m}{n-1}^k$ (with n the number of switches and m the number of links). In a network with k VLANs, we found that each edge replacement in a spanning tree is independent of the $k-1$ other spanning trees. Thus, we reuse and modify the heuristics of LSA4STP in the definition of neighborhood and of move for LSA4MSTP. New heuristic must be defined for the selection of VLAN to perform the move in each iteration.

4.3.1 Algorithm description

Algorithm 6 provides the pseudo-code of LSA4MSTP. At each iteration, we try to replace one edge in one of the k spanning trees to reduce U_{max}. The steps of LSA4MSTP according to the pseudo-code are:

- Lines 1 and 2: The method $getDefaultCosts(BW)$ returns k default link cost matrices W for k VLANs computed from the bandwidth matrix BW. Then k initial spanning trees MSP are computed by the MSTP standard with k link cost matrices W in Line 1. The computation of the $root$ for each VLANs is described in Section 3.3.3 in Chapter 3.

Algorithm 6: Pseudo-code for LSA4MSTP

```
1  $W = getDefaultCosts(BW)$;
2  $MSP = MSTP(G, V, W)$;
3  $U^*_{max} = getMaxUtilization(MSP)$;
4  $MSP^* = MSP$;
5  while $time\_exec < time\_windows$ do
6      $(s_{max}, t_{max}) = getMaxCongestedLink(MSP)$ ;
7      $selected\_vlan = selectVLAN(MSP, s_{max}, t_{max})$ ;
8      $(s_O, t_O) = getRemovedLink(MSP,$
                                   $selected\_vlan, s_{max}, t_{max})$ ;
9      $(s_I, t_I) = getAddedLink(MSP, selected\_vlan, s_O, t_O)$ ;
10     $MSP = replaceEdge(MSP, selected\_vlan, s_O, t_O, s_I, t_I)$ ;
11     $U_{max} = getMaxUtilization(MSP)$;
12     if $U_{max} < U^*_{max}$ then
13         $U^*_{max} = U_{max}$ ;
14         $MSP^* = MSP$;
15     end
16 end
```

- Lines 3 and 4: We store this initial solution obtained with MSTP as the best solution at the start of the search. The method $getMax$-$Utilization(MSP)$ returns U_{max} after computing the utilization of each link in MSP.

- Line 5: In this LS, we also use the time as the termination criterion. The choice of time limits depends on the test size (number of nodes and number of links).

- Lines 6-8: At each search iteration, we first try to find the most congested link (s_{max}, t_{max}) (Line 6). Then, we select a VLAN $selected\_vlan$ that contains (s_{max}, t_{max}) (Line 7). An edge (s_O, t_O) in $selected\_vlan$ will be chosen to be removed (Line 8). We describe this task in Section 4.3.2.

- Line 9: Section 4.3.3 describes the choice of the replacing edge (s_I, t_I) in the method $getAddedLink(MSP, selected\_vlan, s_O, t_O)$.

A new spanning tree for $selected\_vlan$ is created by replacing (s_O, t_O) with (s_I, t_I).

- Line 10: An update of the link loads is performed when the edge (s_O, t_O) is replaced by (s_I, t_I).
- Lines 11-14: If the new U_{max} is lower than the best known U^*_{max}, we store this solution as the best one.

The search process from line 6 to line 14 is iterated until the execution time reaches the time limit (Line 5). As mentioned in 3, the main challenge when dealing with this TE problem for large DCNs is to find good solutions in reasonable time. Thus, we try to find lightweight but efficient heuristics for selecting the edge to be removed and the edge to be added for LSA4MSTP.

4.3.2 Removing an edge

In each search iteration, we try to relieve the most congested link (U_{max}) of its load by replacing an edge in the spanning tree containing it. To determine which edge from which VLAN needs to be replaced, LSA4MSTP extends the heuristic of LSA4STP. Let (s_{max}, t_{max}) be the most congested oriented link, the key decision of this heuristic is to select one of the VLANs containing (s_{max}, t_{max}) to do the replacement. In the method $selectVLAN(MSP, s_{max}, t_{max})$ (Line 7 - Algorithm 6), we create a set SV of VLANs that contains (s_{max}, t_{max}). We assign to each VLAN in SV a probability to be selected based on its load on (s_{max}, t_{max}). The probability for a $vlan \in SV$ to be $selected\_vlan$ is:

$$pr[vlan] = \frac{L_{vlan}[s_{max}, t_{max}]}{\sum_{i=1}^{k} L_i[s_{max}, t_{max}]}. \text{ Obviously, } \sum_{i \in SV} pr[i] = 1.$$

This strategy can find a balance between greedy search and unexplored neighborhood. From $selected\_vlan$, we can assume that the congestion is caused by the traffic coming from the subtree of the spanning tree $SP_{selected\_vlan}$ dominated by s_{max}. Next, the method $getRemovedLink$ $(MSP, elected\_vlan, s_{max}, t_{max})$ (Line 8 - Algorithm 6) uses the heuristic of LSA4STP to determine an edge (s_O, t_O) to be removed from the set of edges containing (s_{max}, t_{max}) and all the edges belonging to the subtree dominated by s_{max}. The edges closer to the root have a higher probability of being removed.

4.3.3 Adding an edge

After having removed (s_O, t_O) from $SP_{selected\_vlan}$, we obtain two separate trees that must be reconnected with a new edge. Our objective is to have a less congested solution. We consider two criteria for choosing an edge to be added to form the new spanning tree. First, we call SA the set of all the edges that join the two separate trees. Second, we consider h edges having the highest remaining bandwidth from SA. Next, we compute the resulting U_{max} when adding each of these h edges. The edge (s_I, t_I) offering the minimal value of U_{max} is selected to be added into $SP_{selected\_vlan}$.

This heuristic differs from LSA4STP as an edge can belong to many spanning trees. Therefore, we can consider only the highest *remaining* bandwidth edges instead of the highest bandwidth edges.

Tabu list

We also use tabu search for preventing the search from replacing the same couple of edges in the same spanning tree. In this problem, we insert only the added edge in each search step into the tabu list. We do not tabu the max congested edge and removed edge as in LSA4STP because in MSTP, an edge can be used by different spanning trees.

4.3.4 Speeding up the search

Besides the root selection technique to eliminate the symmetries and the incremental link load computation to measure U_{max} and to update the load matrix, the data structures of LSA4MSTP are designed to dynamically model multiple spanning trees. In LSA4MSTP, spanning trees are also represented using the $LS(Graph\&Tree)$ framework [DDVH09]. With these incremental data structures (auto-update after each change of the tree), all queries on the spanning tree mentioned above can be performed in time $O(1)$ and the update action is performed in $O(n_v)$ where n_v is the number of vertices of the VLAN v. The computation of U_{max} for each trial edge replacement is performed locally on the $selected\_vlan$. This computation is also based on the "magic cycle" described in Section 3.3.5 in Chapter 3.

4.4 Data Generation

In this section, we first present two topologies coming from the private enterprise and cloud data centers which are studied in [BAM10]. Second, we describe the method for generating these topologies, the traffic demand matrices and the VLANs used for our tests. Next, we analyze the obtained results and evaluate the performance of our local search algorithm LSA4MSTP.

4.4.1 Data Center topologies

The extensive studies by Benson et al. [BAM10] showed that there are three main classes of data centers, namely university campus data centers, private enterprise (PR) data centers and cloud data centers. They collected statistics from 10 data centers in US and South America. In this work, we only consider the large data centers of private enterprises and clouds containing a few thousand to more than 10K servers. We choose to not consider university campus data centers because their size is often too small, containing only a few dozens of switches. As depicted in Fig. 4.1 and Fig. 4.2, private enterprise data centers often use a canonical 2 or 3-Tier Cisco architecture [Sys10] while the cloud data centers often use the 3-Tier data center architecture in [GHJ$^+$09].

3-Tier Cisco architecture

The architecture in Fig. 4.1 consists of core, aggregation and edge (or access) tier. At the highest level, the core tier contains switches connecting the data center to extranet, WAN or Internet. The aggregation tier consists of switches connecting to many edge tier uplinks, and aggregating flows going in and out of the data center. Core and aggregation switches are usually equipped with 10 Gbps interfaces [Sys10]. At the lowest level, edge tier consists of the racks. Each rack contains 20-80 servers interconnected by a Top of Rack switch (ToR). Each ToR switch has usually a small number (4-8 ports) of 10 Gbps uplinks and servers are attached to their ToR switch through 1 Gbps links [Sys10]. The 2-Tier Cisco architecture is used in small data centers where the core tier and aggregation tier are merged into one tier.

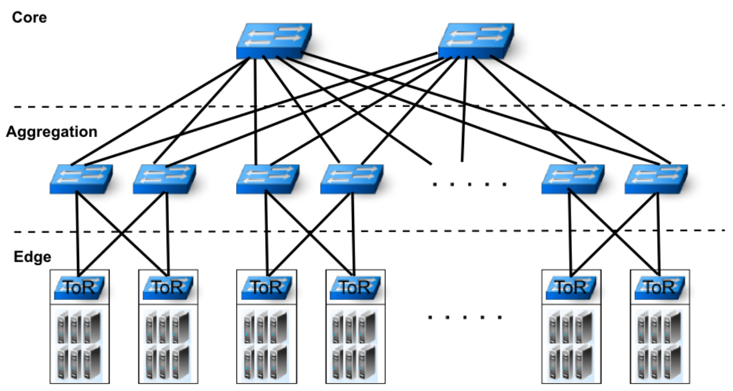


Figure 4.1: Private Enterprise DC

Cloud data center architecture

The cloud data center architecture described in [GHJ+09] (Fig. 4.2) is an improvement of the canonical 3-Tier Cisco architecture. In this topology, the core tier is replaced by an intermedia tier to improve the performance of the aggregation layer. A large number of 10GigE ports of each aggregation switch are used to provide a huge aggregate capacity. The links of the intermediate and aggregation switches form a complete bipartite graph [GHJ+09]. Suppose each aggregation switch uses k 10 GigE ports, $k/2$ of these ports will connect to $k/2$ switches in the intermedia tier. The remaining $k/2$ ports of each aggregation switch are used for connecting to the ToRs in the edge tier. As in 3-Tier Cisco architecture, core and aggregation switches are equipped with 10 Gbps interfaces and ToRs have 4-8 10 Gbps interfaces and a large number of 1Gbps links.

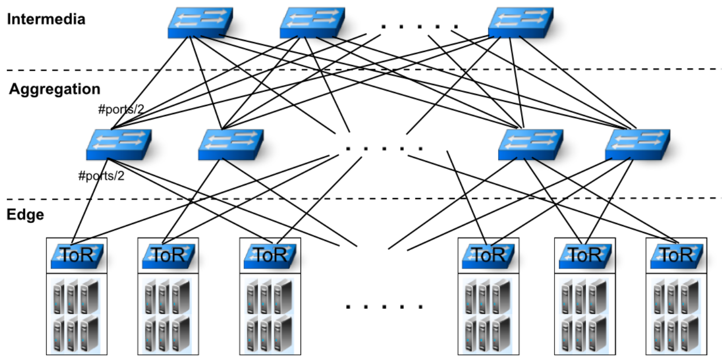


Figure 4.2: Cloud DC

In our experiments, the number of servers per rack is fixed to 20. Thus, each private enterprise data center with 4K servers consists of 242 switches (200 ToRs + 40 aggregation switches + 2 core switches) and each cloud data center with 10K servers contains 564 switches (500 ToRs + 32 aggregation switches + 32 intermedia switches).

4.4.2 Traffic demand matrices and VLANs generation

To obtain data sets that are representative of data centers, we analyzed the SNMP data from [BAM10] on a private enterprise data center (53 switches). This data set allows us to synthesize the traffic flow in the network by every 10 minutes, hour, day and week. Unfortunately, there is no information related to the VLANs composition and on the traffic demand between each pair of switches.

Traffic demand matrices

In spite of the fact that the VLAN and traffic demand information unavailable, the SNMP data from [BAM10] is worthy to infer traffic demand matrices. The SNMP data captures the amount of traffic on each link at a precise time. Therefore, we were able to compute the total traffic entering or departing from each switch over various time intervals.

We use the *simple gravity model* [ZRDG03] relying on proportionality relationships to build the traffic demand matrices. This method was developed for large-scale IP networks but we assume that it is also appropriate for data center networks. The simple gravity model is defined as follows:

$$TD[i,j] = TI(i,*)\frac{TO(*,j)}{\sum_k TI(*,k)}.$$

where $TD[i,j]$ is the traffic that switch i sends to switch j,
$TI(i,*)$ represents the total traffic entering at switch i,
$TO(*,j)$ denotes the total traffic departing at switch j,
and $\sum_k TI(*,k)$ is the total amount of traffic departing of all switches.

In these data centers, there is always a switch receiving a large amount of traffic (20-40$\%$ total traffic). There are about ten other switches receiving from 2 to 18$\%$ of the total traffic. For the remaining switches, the traffic amount is less than 2%. When we look at each line of the traffic demand matrices, each switch has about ten “large clients" with a demand from 2 to 30$\%$ of its total traffic volume.

For each time sample, we thus obtain a demand matrix. For each demand matrix, we define $SumTD = \sum TD[i,j]$, denoting the total amount of traffic demand. We compute the ratio between the traffic volume of each switch and the total traffic: $\%TD\_SW[i] = \dfrac{\sum_j TD[i,j]}{SumTD}$ and the ratio betwwen each traffic demand element and the total traffic demand of each switch: $\%TD[i,j] = \dfrac{TD[i,j]}{TD\_SW[i]}$. The obtained demand matrices will be used to generate the traffic demand matrices of our VLANs. The traffic demand of each of our VLAN will thus be considered as the demand of a small private enterprise data center. For the number of racks (ToRs) in each VLAN, we consider 40 ToRs for each VLAN in cloud topologies (with 564 nodes) and 20 ToRs for each VLAN in private topologies (with 242 nodes). Each VLAN will then also contain the minimum number of aggregation and core/intermediate switches in order to cover the ToRs of the VLAN.

In our experiments, we consider three types of traffic demand matrices for the VLANs.

- **Internal TM**: We here assume that all the traffic stays within the VLAN and consists of flows accross the k racks (ToRs) of the

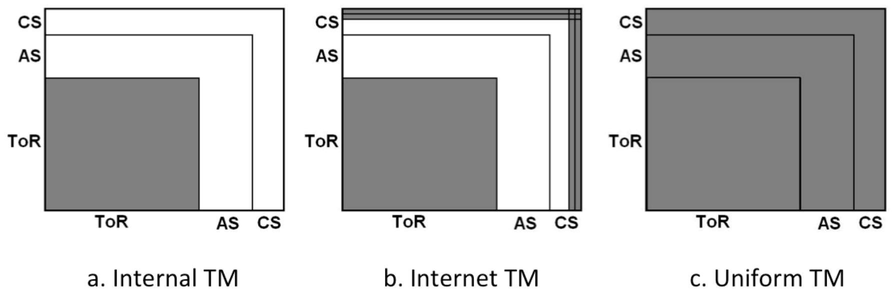


Figure 4.3: Traffic demand matrix generation

VLAN. The demand matrix of the VLAN is thus composed of zeros, except between these k ToRs (Fig. 4.3a). The traffic demand between the ToRs of the VLAN is based on the obtained demand matrices on a private enterprise data center (PR) presented above. We first choose a target $SumTD$. Then, using $\%TD\_SW[i]$ and $\%TD[i,j]$, we derive a demand matrix as follows. The k ToRs of the VLAN are randomly assigned to k different nodes of PR (the nodes of PR with the smallest $\%TD\_SW[i]$ are not considered). Then, given a ToR of the VLAN associated to node i of PR, its traffic demand with the other k ToRs of the VLAN will be a random permutation of the top k values of $\%TD[i,j]$. These values are then randomly assigned.

- **Internet TM**: This case extends the previous case by considering traffic outside network, consisting of traffic accross VLANs and traffic from/to Internet. The traffic entering/leaving each VLAN is centralized at one or two core switches (with an average of 1.5) for private enterprise networks, and at two core switches for cloud data centers. The traffic demand matrix will thus have one or two other non zero lines and columns (as described in Fig. 4.3b), associated to these core switches. We will assume that 20% of $SumTD$ is interconnection traffic, and 80% of $SumTD$ stays within the VLAN. The traffic demand within the VLAN is obtained as described above. The interconnection traffic is uniformly distributed among all the switches.

- **Uniform TM**: This type of traffic will be used as a reference for the experiments, with a uniform distribution of the traffic demand between every pair of switches in the VLAN (see Fig. 4.3c). The values in the matrix are varied in a small interval. The values are chosen to achieving the targeted $SumTD$.

VLANs generation

We rely on two approaches for generating VLANs.

- **Geographic:** each VLAN is generated geographically by grouping a set of neighboring racks that are interconnected by the ToR switches and a number of aggregation and core (or intermedia) switches (i.e. the racks of servers in the same or neighboring buildings).

- **Random** we assume that the racks are assigned randomly to the different VLANs depending on their increasing need. For this reason, each VLAN can contain a set of arbitrary racks.

In our experiments, for each data center topology, we generated 16 VLANs for both geographic and random case. The 16 VLANs are generated by combining the four time samples (4 VLANs generated using each of the time sample). We also ensured that the 16 VLANs cover all the switches.

In order to analyse the influence of the number of VLANs on the performance of our algorithm, we merged the 16 VLANs, 2 by 2, in order to obtain a new test with 8 VLANs, with an equivalent total traffic. Two VLANs can be merged if they have at least one common switch (for the geographic case, this common switch must be a ToR switch). We repeated this process to obtain tests with 4, 2 and a unique VLAN (containing all the switches in the network).

4.5 Experiments and Evaluation

The different test sets are summarized in Table 4.1. For each topology family (private enterprise and cloud), we generated 10 topologies. For each topology, we combined two VLAN distributions (Geographic and

Random) with three traffic matrices (Internal TM, Internet TM and Uniform TM). For each of these 12 combinations, we generated 5 tests (16, 8, 4, 2 and 1 VLAN). We thus have 600 tests in our data sets. These data sets are available online in [HDBF11a]. The time limit for running LSA4MSTP for Private Enterprise and Cloud is 15 minutes.

Evaluation

As with LSA4STP in Chapter 3, we consider the improvement of the maximal utilization U_{max} as the criterion to evaluate the performance of LSA4MSTP. We compare two different computations of U_{max}. The first one is obtained with the solution of LSA4MSTP and the second is obtained with the default link costs of the MSTP 802.1s standard. With 802.1s, one spanning tree is computed for each VLAN based on the least cost path from each switch in this VLAN to an elected root switch (the switch with min ID - normally this is one of the core switches). Because the network contains all 10 Gbps uplinks with the default cost of 2 (see Table. 2.1 in Chapter 2), so the least cost path strategy of 802.1s seems to be limited.

We measure the improvement in each test as the ratio between U_{max} $[LSA4MSTP]$ and $U_{max}[802.1s]$:

$$\%Improve = \frac{U_{max}[LSA4MSTP] * 100}{U_{max}[802.1s]}$$

Fig. 4.4 presents the U_{max} values for Cloud data centers. LSA4MSTP always gives the best results for 16 VLANs, with $\%Improve$ around 50% (about half of the U_{max} given by 802.1s). For 8 VLANs, this improvement is about 60% in both geographic and random case. With 4, 2 and even 1 VLAN, LSA4MSTP also reduces U_{max} to about 70%-80% in almost all the combinations. These results clearly show that our LSA4MSTP algorithm provides better performance than 802.1s.

We describe in Table 4.2 the $\%Improve$ results for Private Enterprise data centers. We also observe that LSA4MSTP is more efficient when the number of VLANs is large. For 16 VLANs with the original traffic matrices, LSA4MSTP gives best performance with the uniform traffic matrices where there is no zero-demand for every source-destination pair. The improvement is less important with the sparser traffic matrices as the internal VLAN matrices. In summary, our LSA4MSTP algorithm always provides better results than 802.1s with the default link costs. Moreover, increasing the number of VLANs clearly further improves the quality of the solution produced by LSA4MSTP.

Fig. 4.5 shows the distribution of the link utilization with the solution provided by LSA4MSTP and 802.1s on the private enterprise topology (Internal TM/Geographic). With 16 VLANs, the 802.1s solution uses only 822 links while LSA4MSTP uses 1008 links (we consider both directions of a link). This distribution shows that the high values of U_{max} in the 802.1s solution are concentrated on few links. The LSA4MSTP solution, by using more links, is able to reduce U_{max} from 0.66 to 0.37. For 8 VLANs, the most congested links are only concentrated on very few links (less than 1%) in the solutions obtained by using the default 802.1s link costs. The link utilization of the other links are similar in the two solutions. This analysis can also be made on the other combinations with 4 and 2 VLANs where the congestion in the solutions obtained with 802.1s is centralized in about 2 or 3 bottleneck links.

We now analyze the influence of the number of VLANs on the number of used links. Let $\#Links[LSA4MSTP]$ and $\#Links[802.1s]$ denote the number of links given by the solution of LSA4MSTP and of 802.1s. We compute the percentage of links on the total number of available links $\#Links$ in the network that LSA4MSTP uses more than 802.1s:

$$\%\Delta Links = \frac{(\#Links[LSA4MSTP] - \#Links[802.1s]) * 100}{\#Links}$$

We present in Table 4.3 the value of $\%\Delta Links$ for each topology type. In the solutions given by LSA4MSTP, the spanning trees use more links than the ones obtained with 802.1s for all the combinations with more than 1 VLAN where $\#Links[LSA4MSTP]$ and $\#Links[802.1s]$ are fixed to $n-1$ (with n the number of switches in the data center). We can thus disjoin the VLAN spanning trees on the most congested links. The value of $\%\Delta Links$ increases naturally with the number of VLANs. This justifies why the best U_{max} results are obtained with 16 VLANs.

We further refine our analysis by presenting in Table 4.4 the average number of links connecting Intermedia - Aggregation (Int-AS) and

Aggregation - Edge (AS-ToR) for Cloud data centers. For both 802.1s and LSA4MSTP solutions, the number of used links Int-AS is very limited (always less than 100 links) comparing to the available links on this level (1024 links). In contrast, the number of AS-ToR links is growing quickly with the number of VLANs (up to 71% of available links on this level). Obviously, for the Cloud data centers there are 500 ToRs and only 32 Aggregation switches + 32 Intermedia switches. In addition, 80% of the total traffic amount is used for the traffic across racks. Our LSA4MSTP algorithm always uses more links than 802.1s. It is interesting to notice that LSA4MSTP can reduce 50% of U_{max} with only 63 more links for 16 VLANs.

We finally describe in Figure 4.6 the improvement of the LSA4MSTP solution over execution time, for a test of the Cloud with 16 VLANs/Uniform TM/Geographic. As expected, LSA4MSTP reduces about 50% U_{max} of 802.1s (from 0.69 to 0.33) after only 10s. We can state that most of improved solutions were found in the first 98s. In our experiments, the solution found by LSA4MSTP in the first 5 minutes is often very close to the best solution.

With the obtained results in our previous work with LSA4STP with Grid, Cube, Expanded Tree, Fat Tree and PortLand, our local search algorithms give good performance with large instances of network topologies.

4.6 Conclusion

In this chapter, we gave a new approach for the minimization of U_{max} in the DCNs where the MSTP 802.1s is deployed. To cope with large DCNs containing many VLANs, the LS algorithm proposed in Chapter 3 has been extended with new heuristics. We considered the current modern topologies for large data centers containing up to 10K servers in our experiments. The SNMP data of a private enterprise in the US has also been used to create traffic demand matrices for our tests. With regard to the load balancing aspect, our results show much improvement in the use of network available bandwidth. The solutions obtained with our LS algorithm could reduce up to 50% the maximal link utilization compared with the solution obtained by 802.1s for the DCNs with 16 VLANs. In next chapter, besides U_{max}, we take into account the service disruption in case of link failures as the second objective to be optimized.

Table 4.1: Data generation for LSA4MSTP

Name	Type	# Switches	# Servers	Traffic Matrices	VLAN distribution
Private Enterprise (PR)	3-Tier Cisco	242	4,000	Internal TM	Geographic/Random
Private Enterprise (PR)	3-Tier Cisco	242	4,000	Internet TM	Geographic/Random
Private Enterprise (PR)	3-Tier Cisco	242	4,000	Uniform TM	Geographic/Random
Cloud	VL2	564	10,000	Internal TM	Geographic/Random
Cloud	VL2	564	10,000	Internet TM	Geographic/Random
Cloud	VL2	564	10,000	Uniform TM	Geographic/Random

Table 4.2: Results for Private Enterprise data centers: $\%Improve$ (in percent)

Combination	1 VL	2 VLs	4 VLs	8 VLs	16 VLs
Geographic/Internal TM	85.47	83.20	70.42	54.30	52.11
Geographic/Internet TM	87.67	83.04	73.67	52.43	51.26
Geographic/Uniform TM	88.97	83.05	80.51	62.13	42.46
Random/Internal TM	84.15	83.33	69.91	61.92	57.60
Random/Internet TM	83.36	75.63	70.25	58.54	52.19
Random/Uniform TM	88.03	80.77	78.17	62.85	42.02

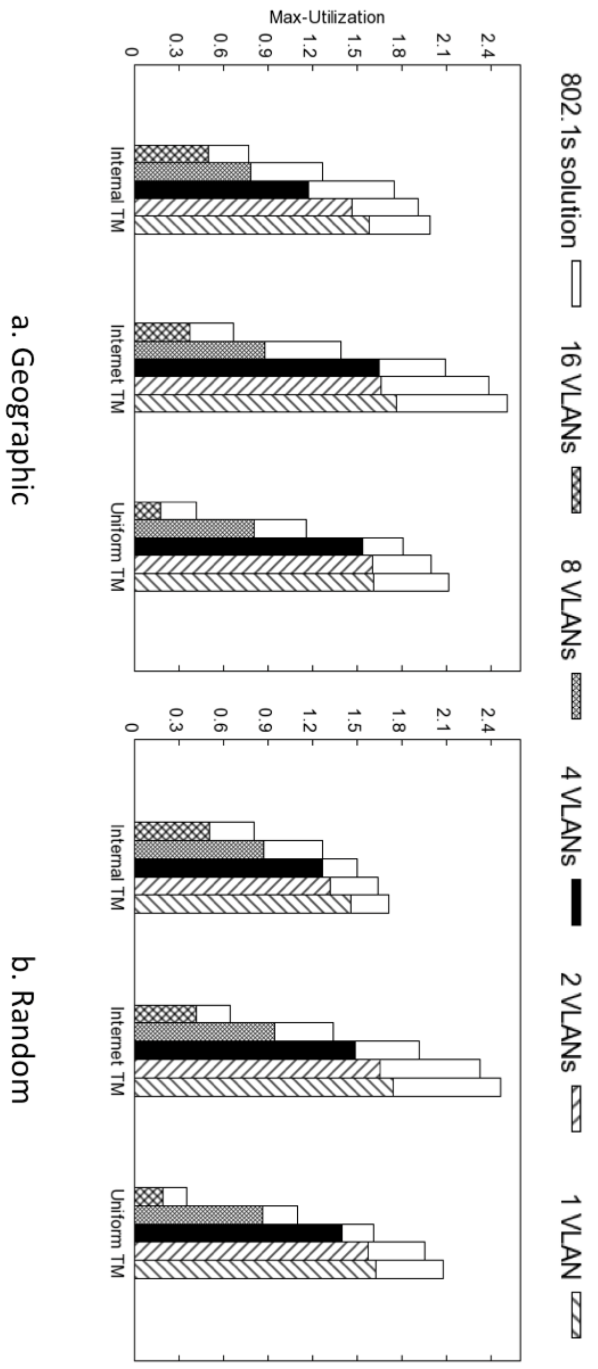


Figure 4.4: LSA4MSTP for Cloud DCs

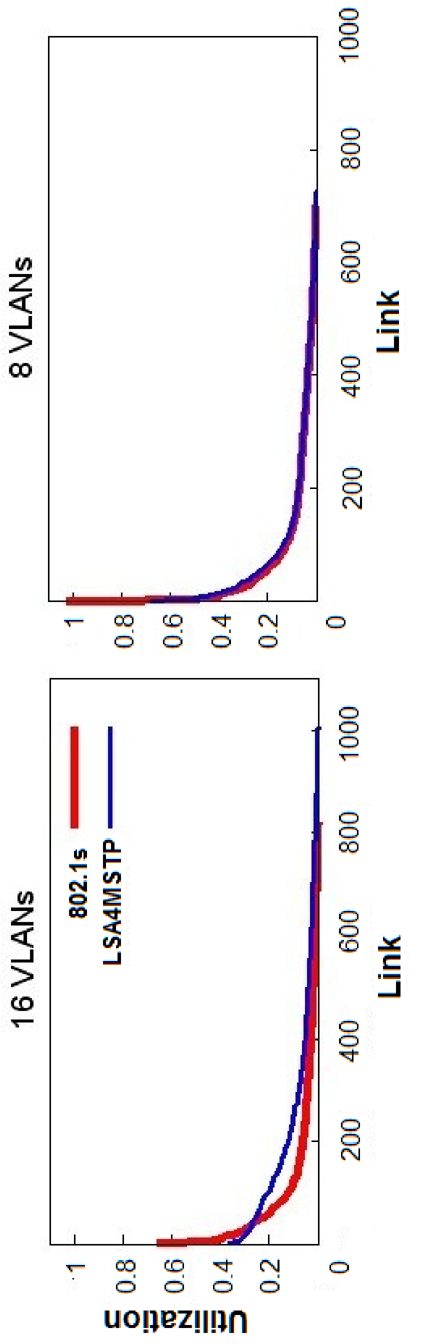


Figure 4.5: Link utilization distribution

Table 4.3: $\%\Delta Links$ between LSA4MSTP and 802.1s (in percent)

Topo	1 VLAN	2 VLANs	4 VLANs	8 VLANs	16 VLANs
PR	0	2.59	7.26	10.46	12.54
Cloud	0	0.71	2.05	2.79	3.41

Table 4.4: Number of used links across tiers for Cloud data centers (LSA stands for LSA4MSTP)

Links	Available	1 VLAN		2 VLANs		4 VLANs		8 VLANs		16 VLANs	
		802.1s	LSA	802.1s	LSA	802.1s	LSA	802.1s	LSA	802.1s	LSA
Int-AS	1024	63	63	66	69	73	78	75	81	91	98
AS-ToR	1000	500	500	521	532	536	573	598	648	656	712
Total	2024	563	563	587	601	609	651	673	729	747	810

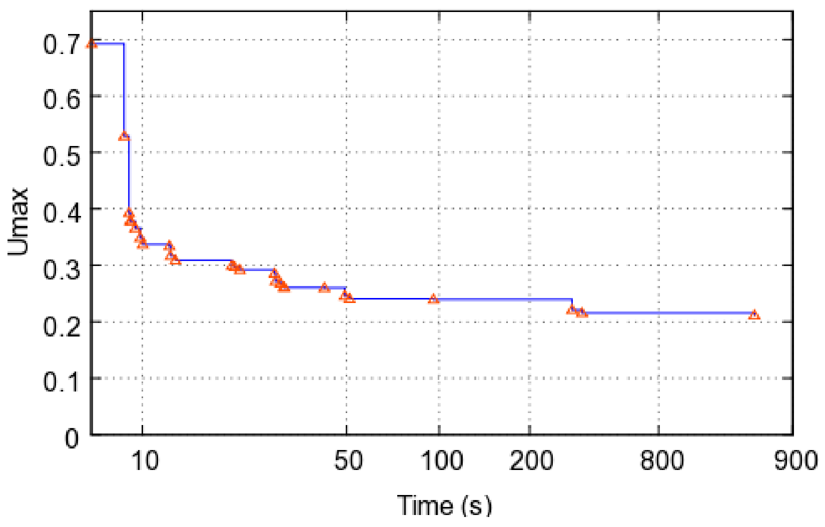


Figure 4.6: Improvement of U_{max} over execution time

5 DEALING WITH LINK FAILURES

In this chapter, we introduce a heuristic to ensure a good load balancing for the DCNs with a minimum service disruption in case of link failure. We show how to configure the link costs such that the MSTP standard performs a single link replacement when one of its spanning tree links fails. This minimum service disruption is considered as a hard constraint in our heuristic for minimizing the worst-case maximal link utilization in the DCNs.

5.1 Introduction

Besides the ultimate objective of congestion avoidance, a natural expectation of network operators is to minimize the convergence time for recovering connectivity and the number of traffic flows that suffer from service disruption in case of link failures. In DCNs, the minimization of the service disruption can be expressed as minimizing the number of link replacements in each VLAN to maintain the spanning tree after failures. In addition, it is difficult to perform permanently failure checks and give out immediately a new good set of spanning trees. Thus a good configuration of link costs is required to guide the MSTP ensuring the network performance in both cases: with or without failure.

In dealing with this TE problem, we take into account two objectives: minimization of the worst-case maximal link utilization and minimization of the number of link replacements in each VLAN considering each single link failure from a set of given links. Obviously, the minimal number of link replacements for each spanning tree after a single link failure is one. Thus, the minimization of the number of link replacements can be considered as a hard constraint for the minimization of the worst-case maximal link utilization. Assume that our DCN has m used links and k VLANs, with each failure in these m links, MSTP must generate a set of k spanning trees. Each solution is then composed of $m \times k$ spanning trees. In addition, these spanning trees must be followed by a generation of link costs for each VLAN. Therefore, the computation of even an arbitrary solution for this problem in DCNs with thousands of links is very costly.

The recent study on network failures in DCNs by Gill et al. in [GJN11] shows that the failure of the links from the Edge (ToR) level often has smaller impact on data transmission than the links in the higher levels (Aggregation and Core) because they are the least used. Furthermore, the links from the Edge level are more reliable and less sensible to failures than the other links in the network. In a spanning tree topology the links from the Edge level are often connected to the leaf nodes, hence their failure has lower impact than the other ones. From the network traffic viewpoint, the failure of a link with higher utilization is more serious than a link with lower one. To reduce the complexity of the problem, we only consider the failure of the set of links with the highest utilization instead of considering all the used links in the network.

Related Work

Many existing techniques have been proposed to deal with the minimization of the maximal link utilization U_{max} in the Ethernet networks (see Section “Related Work" in Chapter 4). But there is no previous work dealing with the minimization of the worst-case maximal link utilization after link failures. In [dSS07], the authors propose two algorithms dealing with the TE problem in Ethernet networks implementing MSTP: (1) A GRASP (Greedy Randomized Adaptive Search Procedure) algorithm aims to minimize U_{max} by generating a set of spanning trees and assigning them to a set of given VLANs; (2) A PortCost assignment algorithm that computes the link cost matrices for the best set of spanning trees found by the GRASP algorithm in (1) ensuring the minimum service disruption in the network due to the failure of each active link in the network. The drawback of these two algorithms is that the GRASP algorithm does not take the link failures into account while the PortCost assigment in (2) does not care about the worst-case maximal link utilization due to link failures. In addition, these techniques are supposed to deal with a small size Ethernet network with 23 switches and 42 links. In this work, our optimization scheme aims to minimize the worst-case maximal link utilization with a minimum service disruption due to link failures in DCNs.

5.2 Problem Statement

The Ethernet network is modeled as described in Section 4.2 in Chapter 4 with:

- Undirected graph $G = (N, E)$ where N is the set of switches and E is the set of links between switches.
- BW is the matrix of link bandwidths.
- $V = \{V_1, V_2, \dots, V_k\}$ is the set of k given VLANs in the network, with $V_r = (N_r, E_r)(N_r \subseteq N,\ E_r = \{(i,j) \in E | i,j \in V_r\}$, $1 \leq r \leq k$).
- $TD = \{TD_1, TD_2, \dots, TD_k\}$ is the set of k traffic demand matrices where $TD_r[i,j]$ represents the traffic that switch i sends to switch j in VLAN V_r ($1 \leq r \leq k$; $i, j \in V_r$).
- $W = \{W_1, W_2, \dots, W_k\}$ is the set of link cost matrices for the k given VLANs and $MSTP(G, V, W)$ is the set of k spanning trees $SP_1(G, V_1, W_1)$, $SP_2(G, V_2, W_2)$,..., $SP_k(G, V_k, W_k)$: In this problem, the link cost matrices W and the k spanning trees are given as input.

The Ethernet switching problem is always to distribute the traffic demands $TD_r[i,j] > 0$ ($1 \leq r \leq k$; $i, j \in V_r$) over unique path from i to j in $SP_r(G, V_r, W_r)$.

We call $used\_links$ the set of all the links that are used by the k spanning trees:

$$used\_links = \{e | e \in E \ \& \ \exists i : e \in SP_i\} \text{ with } i \in [1..k]$$

Let $fail\_links \subset used\_links$ be the set of links for which we must consider the failure. Besides the link load L, link utilization U and U_{max}, in this work, we introduce the concept of the worst-case maximal link utilization U_{fails}. Let $U_{max}[i]$ be the U_{max} computed on a given set of k spanning trees $SP'_{i,1}, SP'_{i,2}, \ldots, SP'_{i,k}$ after the failure of some link $e_i \in fail\_links$ ($\forall j \in [1..k], e_i \notin SP'_{i,j}$). We compute U_{fails} as follows:

$$U_{fails} = max\{U_{max}[i] | \forall e_i \in fail\_links\}$$

Our problem formulation for this TE problem is defined as follows:

PROBLEM 3
Input: Graph $G = (N, E)$, set of k VLANs V, bandwidth matrix BW, set of k traffic demand matrices TD, k link cost matrices W, k initial spanning trees $MSP = \{SP_1, SP_2, \ldots, SP_k\} (= MSTP(G, V, W))$, set of considered links $fail\_links$.

Output: A set of k link cost matrices W^* minimizing the worst-case maximal link utilization U_{fails} w.r.t:

- $MSTP(G, V, W^*)$ generates k spanning trees $SP_1, SP_2, ..., SP_k$ if there is no link failure in the network.
- For each failure of link $e_i \in fail\_links$, $MSTP(G, V, W^*)$ generates k spanning trees $SP^*_{i,1}, SP^*_{i,2}, \ldots, SP^*_{i,k}$ such that:
  - $SP^*_{i,j} = SP_j, \forall j \in [1..k]$ and $e_i \notin SP_j$
  - $SP^*_{i,j}$ is obtained from SP_j by replacing e_i with some $e_h \in E_j, \forall j \in [1..k]$ and $e_i \in SP_j$

We impose to replace the failed link by a single link in each spanning tree that contains it. This can be seen as a hard constraint.

5.3 Heuristic Algorithm for MSTP in case of link failures

In this section, we present our heuristic algorithm called HA4MSTP for minimizing the worst-case maximal link utilization U_{fails} in case of link failures ensuring a minimum service disruption. Algorithm 7 provides the pseudo-code of HA4MSTP. We iteratively select each link in $fail\_links$ to measure the impact of its failure and to configure the link cost matrices in order to guide the MSTP standard to perform a good link replacement in each affected VLAN. The steps of HA4MSTP according to the pseudo-code are:

Algorithm 7: Pseudo-code for HA4MSTP

```
1  $W^* = W$;
2  $U_{fails} = 0$;
3  while $fail\_links \neq \emptyset$ do
4      $e_O = getMostUtilizedLink(fail\_links)$;
5      $vset = getVlansOnLink(e_O)$;
6      Sort $vset$ decreasing by load on $e_O$;
7      $restBW = getRestBW(MSP)$;
8      foreach $vl \in vset$ do
9          $e_I = getReplacingLink(e_O, V_{vl}, restBW)$;
10         $W^*_{vl} = updateLinkCost(V_{vl}, e_O, e_I)$;
11         $restBW = updateRestBW(e_O, e_I)$;
12     end
13     $U_{max} = computeUmax(W^*)$;
14     if $U_{max} > U_{fails}$ then
15         $U_{fails} = U_{max}$;
16     end
17     $fail\_links = fail\_links \setminus \{e_O\}$;
18 end
```

- Lines 1-2: We initialize W^* by copying the link cost matrices of W. U_{fails} is also intialized with a value of 0.

- Line 3: The failure of each link in $fail\_links$ is considered step by step by HA4MSTP. The algorithm finishes when all links in $fail\_links$ have been considered.

- Line 4: The method $getMostUtilizedLink(fail\_links)$ returns the most utilized (sum of its utilization on both two directions) link e_O in $fail\_links$. Our heuristic in this step is to consider the links in $fail\_links$ by a decreasing order of its impact in the network. In each VLAN, the link cost update for selecting a back-up link for a given link can affect the choices of back-up link for other links (see Section 5.4). Thus, this heuristic allows us to guarantee the quality of the back-up link for the most important links in the network.

- Lines 5-8: For each selected link e_O, the set of VLANs $vset$ that contains e_O in their spanning tree is computed by method get-$VlansOnLink(e_O)$ (Line 5). Although an edge replacement in the spanning tree of each VLAN is independent from the others, but it influences the traffic in the whole network. By sorting and considering the VLANs in $vset$ decreasing by their load on e_O (Line 6 and Line 8), we first find an issue for the VLANs that charged the most e_O. The method $getRestBW(MSP)$ in Line 7 returns the matrix of remaining bandwidths $restBW$ of each link in the network before the failure of e_O.

- Line 9: The method $getReplacingLink(e_O, V_{vl}, restBW)$ returns the back-up link e_I for the failure of e_O in each VLAN $vl \in vset$. The heuristic for selecting e_I is detailed in Section 5.4.

- Lines 10-11: The link cost matrix W_{vl} for VLAN vl is updated to ensure that MSTP does a unique edge replacement (remove e_O and add e_I to the spanning tree) after the failure of e_O (Line 10). Our link cost update technique is described in Section 5.5. In Line 11, $restBW$ is updated by computing the new link utilization matrix after replacing e_O by e_I in VLAN vl.

- Line 13: The maximal link utilization U_{max} is computed after having considered the failure of e_O in all the VLANs in $vset$.

- Lines 14-16: If U_{max} is larger than U_{fails}, we store it as the worst-case maximal link utilization.
- Line 17: e_O is removed from the set of links to be considered $fail\_links$.

5.4 Dependence of Back-up links on Link Cost Configuration

In each VLAN, the link cost matrix cannot be configured such that each link in the spanning tree can choose any link that maintains the spanning tree as the back-up link for its failure.

Example. In Fig. 5.1, (6,2) is chosen as the back-up link for (3,4). To update the link cost matrix, the cost of (1,2), (2,3), (3,4), (4,5), (5,6) and (6,7) is always fixed to 1 because these links belong to the spanning tree. (6,2) is chosen as the back-up link for (3,4) so its cost is also fixed to 5. Then we must configure the other link costs such that when the failure of link (3,4) occurs, only one link replacement (3,4) by (6,2) is performed to maintain the spanning tree and the unique shortest path to root (2) from 4, 5, 6 or 7 is via (6,2). This is represented by the following inequalities:

- $W[4,2] > W[4,5] + W[5,6] + W[6,2]$ (1)
- $W[5,2] > W[5,6] + W[6,2]$ (2)
- $W[7,2] > W[7,6] + W[6,2]$ (3)
- $W[7,1] + W[1,2] > W[7,6] + W[6,2]$ (4)

But what if link (4,5) wants to set (5,2) as the back-up link for its failure? Obviously, the shortest path from 5 to root (2) after the failure of (4,5) is $5 \rightarrow 2$ and thus $W[5,2] < W[5,6] + W[6,2]$. This is contradictory with the inequality (2). Similarly, (4,5) can not set (4,2), (7,2) or (7,1) as the back-up link for its failure. When (6,2) is chosen as the back-up link for (3,4), it must be also the back-up link for (4,5) and (5,6).

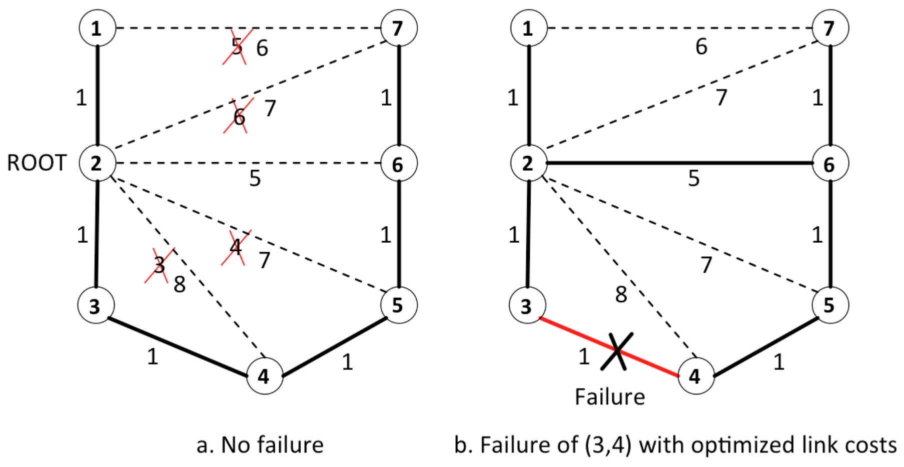


Figure 5.1: STP reconfiguration with optimized link costs

The PortCost assigment algorithm in [dSS07] presents only the mathematical formulas for computing the link costs in each VLAN. It does not provide the dependence of the link cost update on the ordering of the link failures to be considered. In this work, when a link e_I is chosen as the back-up link for link e_O in the spanning tree, we determine the set of links in the spanning tree that must use e_I as their back-up link.

Assume that e_I is chosen as the back-up link of e_O as depicted in Fig. 5.2. Let $C = \{C_1, C_2, .., C_q\}$ be the switches on the cycle created by adding e_I into the spanning tree with $e_O = (C_o, C_{o+1})$, $e_I = (C_q, C_1)$ $(1 \leq o < q)$.

Property 1. *If $e_I = (C_q, C_1)$ is chosen as the back-up link for $e_O = (C_o, C_{o+1})$ then each link (C_{o+1}, C_{o+2}), ..., (C_{q-1}, C_q) must set e_I as the back-up link to guarantee the minimum link replacement for its failure.*

The proof of this property is presented in Appendix B. In Fig. 5.2, the links that must use e_I as their back-up link are in the “must" part of the cycle C_{o+1}, ..., C_q and the links in the “optinal" part C_1, ..., C_o can consider e_I as a candidate for their back-up link.

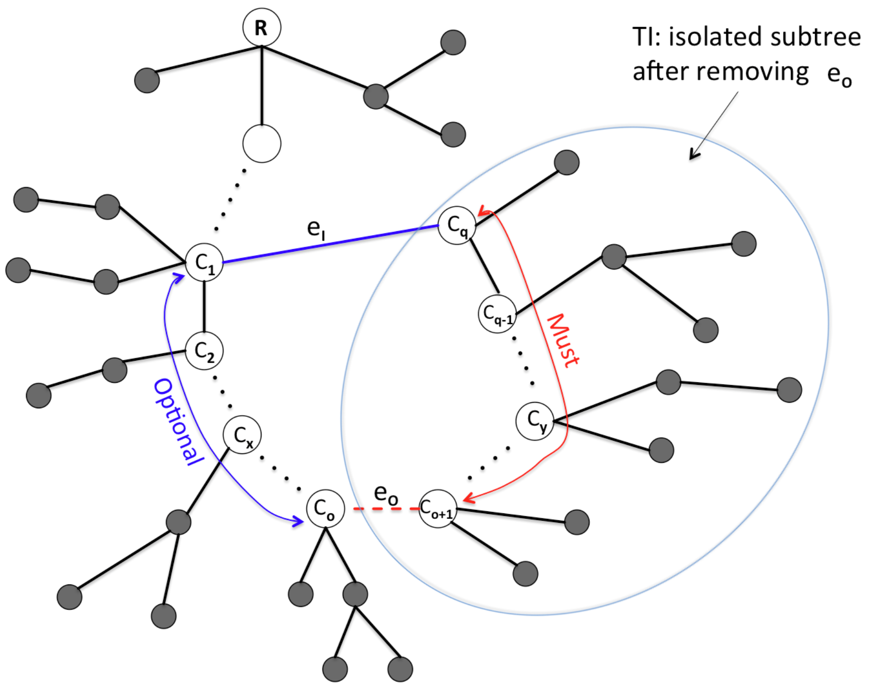


Figure 5.2: Link Cost Update

Heuristic for selecting the back-up link

From this property, our heuristic for selecting the back-up link e_I for the failure of a given link e_O in the method $getReplacingLink(e_O, V_{vl}, rest\text{-}BW)$ (Line 9 - Algorithm 7) is described as follows:

- If e_O is not forced to set any back-up link then we a priori select the candidate replacing link e_I in VLAN V_{vl} with maximal sum of its two-direction remaning bandwidth as the back-up link for e_O. Otherwise, the back-up link e_I is considered for the failure of e_O.

- We compute the cycle $C = \{C_1, C_2, .., C_q\}$ by adding $e_I = (C_q, C_1)$ to the spanning tree SP_{vl} of VLAN V_{vl}. Then e_I is set as the back-up link for each link e in the part of the cycle C_{o+1}, ..., C_q (with $e_O = (C_o, C_{o+1})$ and $1 \leq o < q$).

This heuristic aims to provide the best back-up link (with maximal bandwith) for the most utilized links in the network. In general, these links are also more sensitive to failure.

5.5 Link Cost Update Technique

In this section, we extend the PortCost assigment algorithm in [dSS07] for updating the link cost configuration of each VLAN V_{vl} by considering the failure of a given link e_O in SP_{vl} and its back-up link e_I. In spite of the fact that the input of this problem consists of a network containing many VLANs (one spanning tree for each VLAN), ***the link cost configuration for each VLAN is independent from the other VLANs***.

Property 2. *To maintain the minimum service disruption, a link e_I replaces link e_O in the spanning tree SP_{vl} of VLAN $V_{vl} = (N_{vl}, E_{vl})$ when the failure of e_O occurs if and only if for each switch $v \in N_{vl}$, if the shortest cost path from v to root contains e_O, then its* ***unique*** *second shortest cost path to root must pass via e_I.*

The proof of this property is very simple. Obviously, if there exists a switch $v \in N_{vl}$ that has the shortest cost path from v to root containing e_O and the second shortest cost path to root not via e_I then v will use a different link than e_I to join the root when the failure of e_O occurs. Thus the number of link replacements after the failure of e_O is more than one. Therefore, the minimum service disruption constraint is violated.

The method $updateLinkCost(V_{vl}, e_O, e_I)$ in line 10, Algorithm 7 updates the link cost matrix for VLAN V_{vl} to guarantee that MSTP perform a unique edge replacement (e_O by e_I) on SP_{vl} after the failure of e_O. Let TI be the isolated subtree (excluding root) after the failure of e_O (see Fig. 5.2), the link cost update must ensure that:

- When there are no failures, MSTP computes exactly SP_{vl}
- After the failure of e_O, MSTP generates new spanning tree $SP'_{vl} = SP_{vl} \setminus \{e_O\} \bigcup \{e_I\}$. This means that the unique shortest cost path of each switch in TI must pass via e_I after the failure

of e_O. Hence, only the cost of the links that can join the two separate trees after the failure of e_O (and $\neq e_I$) should be updated (increased).

Algorithm 8: Pseudo-code for Link Cost Update

```
1 $replacing\_edges = getReplacingEdges(e_O)$ ;
2 $replacing\_edges = replacing\_edges \setminus \{e_I\}$ ;
3 $length = getNewCostToRoot(V_{vl}, e_O, e_I)$ ;
4 foreach $e = (i, j) \in replacing\_edges$ do
5     $minCost = length[j] - length[i] + 1$;
6     if $W^*_{vl}[i, j] < minCost$ then
7         $W^*_{vl}[i, j] = W^*_{vl}[j, i] = minCost$;
8     end
9 end
```

The cost update for each candidate back-up link for the failure of e_O is described in Algorithm 8 with the following steps:

- Lines 1-2: We compute $replacing\_edges$ - the set of candiate back-up links that can join the two separate trees after the failure of e_O. When e_I is removed from $replacing\_edges$, each link in $replacing\_edges$ is considered to perform a cost update.

- Line 3: The method $getNewCostToRoot(V_{vl}, e_O, e_I)$ returns the new cost $length[i]$ from each switch $i \in TI$ to root after the failure of e_O (if $i \notin TI$, obviously, its cost to the root does not change).

- Lines 4-7: For each link $e = (i, j) \in replacing\_edges$, it is clear that i and j belong to two different sub-trees after the failure of e_O. We assume without loss of generality that $i \in TI$ and $j \notin TI$. We then compute the minimal cost $minCost$ of e to ensure that the shortest cost path to root from each switch in the subtree dominated by i must pass via e_I (Line 5). If the current cost of e (both directions) is less than $minCost$, we assign $minCost$ as its new cost (Line 6-7).

5.6 Data Set Composition

The seven data center topologies for testing our algorithms are described in Table 5.1. The Private Enterprise topology is a 3-Tier Cisco architecture [Sys10] while the cloud data center uses the 3-Tier textbook data center architecture in [GHJ+09]. In Table 5.1, the two VLAN generation types Geographic (Geo) and Random (Ran) are used to create different tests on the same network topology (same nodes, same edges, but different VLAN generations). The details of Private (PR) and Cloud (CL) architectures are presented in Section 4.4 in Chapter 4.

Table 5.1: Data Set Composition

Size\Type	PR Geo\Ran	CL Geo\Ran	FT Ran	PL Ran	ET Ran
Num. Nodes	242	564	320	313	200
Num. Links	549	2024	2048	602	398

We also use the data center topologies Fat Tree (FT) [AFLV08], PortLand (PL) [NMPF+09] and the generic topology Expanded Tree (ET) as described in Section 3.5 in Chapter 3. To simplify the observation with five different assessment metrics, we generate 16 VLANs and a near-uniform traffic demand matrix for each VLAN for every topology type in our experiments. For each test, we have two types of input link cost matrices. The first one consists of the default costs of MSTP 802.1s and the second one consists of the costs obtained with our LSA4MSTP algorithm for minimizing U_{max} in Chapter 4.

Table 5.2: Results with LSA4MSTP optimized costs (10% links with max. U_{fails})

Topo.	$U_{nofails}$	$\#UL$	U_{fails}		$\sum Changes$	
			802.1s	HA	802.1s	HA
PR Geo.	0.2492	1624	0.4151	0.3388 ($\downarrow$18%)	643 (40%)	16
PR Ran.	0.1901	1752	0.3169	0.2765 ($\downarrow$13%)	815 (47%)	16
CL Geo.	0.2398	3875	0.5524	0.3318 ($\downarrow$40%)	1836 (47%)	16
CL Ran.	0.2673	3730	0.6198	0.381 ($\downarrow$39%)	1738 (47%)	16

5.7 Experiments and Results

In this section, we use the worst-case maximal link utilization U_{fails} in case of link failures as the criterion to evaluate the performance of HA4MSTP. We compare two different analyse of the maximal link utilization in case of link failures: we compute its maximum (U_{fails}-the most important evaluation criterion), its average and the minimum of each test. The first one is obtained with the cost matrices generated by HA4MSTP. The second one is obtained with the input cost matrices. In each test, we consider the failures of 10% of the links in the network that have the highest utilization. These links are also the set of links that are sensitive to failure in the network. We then simulate the failure of each of these links and compute U_{max} on the new spanning tree computed by MSTP 802.1s after failure. The worst-case maximal link utilization U_{fails} is the largest U_{max} among the failures of each of these links. In addition, we also analyze the number of link replacements in case of link failures on the default solutions given by 802.1s to have a general image of service disruption without minimization. With HA4MSTP, the service disruption in case of link failures is minimized with one unique link replacement in the spanning tree of each VLAN.

Fig. 5.3 represents the maximum (U_{fails}), the average and the minimum U_{max} in case of link failures for the input costs and the optimized costs obtained with HA4MSTP. The black points present the U_{max} without link failures. Here, the input costs are assigned by the default link costs of STP 802.1d [Soc98]. HA4MSTP gives the best results for the tests of Private Enterprise (Geo. and Ran.) when its optimized cost matrices guide MSTP to reduces about 20% U_{fails} comparing to the default input costs. For all these 7 topologies, the average reduction is about 9.5%. Additionally, the average and the minimal U_{max} after failure given by HA4MSTP are also smaller than the ones with the default input costs in almost all tests.

In Fig. 5.4, we perform the same observation as in Fig. 5.3 but the input costs are generated by our LSA4MSTP algorithm for minimizing U_{max} when there are no failures. This explains why the solutions with no failure (the black points) are always the best solutions (with minimal U_{max}) for every test. With these tests, HA4MSTP performs better with the large networks when its optimized costs reduces up to about 33%

U_{fails} in comparison to the input costs for the tests of Cloud (Geo. and Ran.). The average reduction for all 7 topologies is about 16.5% and it has better average and minimal U_{max} after failure than the input costs for all 7 topologies in spite of the fact that HA4MSTP performs only one link replacement in the spanning tree of each VLAN after the failure of each considered link.

From the service disruption viewpoint, we present in Table 5.3 the analysis of the number of link replacements after each link failure in each spanning tree. These are the same experiments as depicted in Fig. 5.3. We can state that although the average number of link replacements after each link failure $avgChg$ of the solutions obtained with the default input costs for all 7 topologies is just around 2 and its minimal $minChg$ is 1 (optimized) but its maximum $maxChg$ is from 17 to 42 link replacements (change of up to 40% links in the old spanning after only one link failure). For this objective, HA4MSTP ensures an optimal service disruption with one unique link replacement in the spanning tree of each VLAN after link failure. The execution time for the simulation of MSTP 802.1s is always larger than the one given by HA4MSTP for all 7 topologies.

We can find a similar analysis of the number of link replacements after each link failure but with the LSA4MSTP costs in Table 5.4. In this case, with the non-configured input cost, the spanning tree in each VLAN is almost out of control after each link failure when the $avgChg$ of Fat Tree test is 10.96 and the $maxChg$ of Cloud Ran. climbs up to 130 link replacements after only one link failure. In addtion, the worst-case maximal link utilization U_{fails} in case of link failures obtained with the solutions with the input costs is worse than the ones obtained with HA4MSTP for all given tests. Once more, HA4MSTP is faster in execution time than the simulation of MSTP 802.1s for all 7 topologies.

Last, we consider the failure of 10% of the links in the network that have the highest U_{fails} given by MSTP 802.1s for better understanding the impact of link failures in DCNs. This experimentation is performed on the same tests as Private Geo.\Ran. and Cloud Geo.\Ran. in Table 5.4. The results are presented in Table 5.2 where we measure $U_{nofails}$ - the value of U_{max} in case of no failures (same value as in Table 5.4), $\#UL$ - the number of links that are used by all the 16 spanning trees in case of no failures, U_{fails} and $\sum Changes$ - the total number of link replacements in the solutions obtained with HA4MSTP and with MSTP 802.1s. We observe that in the worst case with the solutions given by MSTP 802.1s, one single link failure can increase up to 230% $U_{nofails}$ (Cloud Ran.: from 0.2673 to 0.6198) and can change up to 47% the spanning tree links (Cloud Ran. 1738/3730, on average 109 link replacements/VLAN). In Table 5.2, we also observe that HA4MSTP can reduce up to 40% U_{fails} (with Cloud Geo.) obtained with MSTP 802.1s by performing exactly one link replacement in each spanning tree: $\sum Changes$ equals to 16 (16 VLANs and 1 replacement/VLAN) for all the tests.

5.8 Conclusion

In this chapter, we introduced a bi-objective TE problem with the minimization of the worst-case maximal link utilization and of the service disruption in case of link failures. Besides the traditional objective of minimizing the congestion, we proposed an efficient link cost configuration technique for each VLAN in the network that can give MSTP an auto-configurable solution with one unique link replacement in the spanning tree of each VLAN after a link failure. Hence, the solution obtained with our heuristic algorithm guarantees the optimum for the minimization of the service disruption and also shows good performance for minimizing the worst-case maximal link utilization in the network in case of link failures.

In this particular TE problem, the optimum is always guaranteed for the second objective because it can be considered as a hard constraint for the minimization of the worst-case maximal link utilization. In next chapter, we will deal with the bi-objective TE problems where the optimum cannot be guaranteed for any objective and the outcome of our bi-objective algorithms is a set of non-dominated solutions instead of a unique solution as in this chapter.

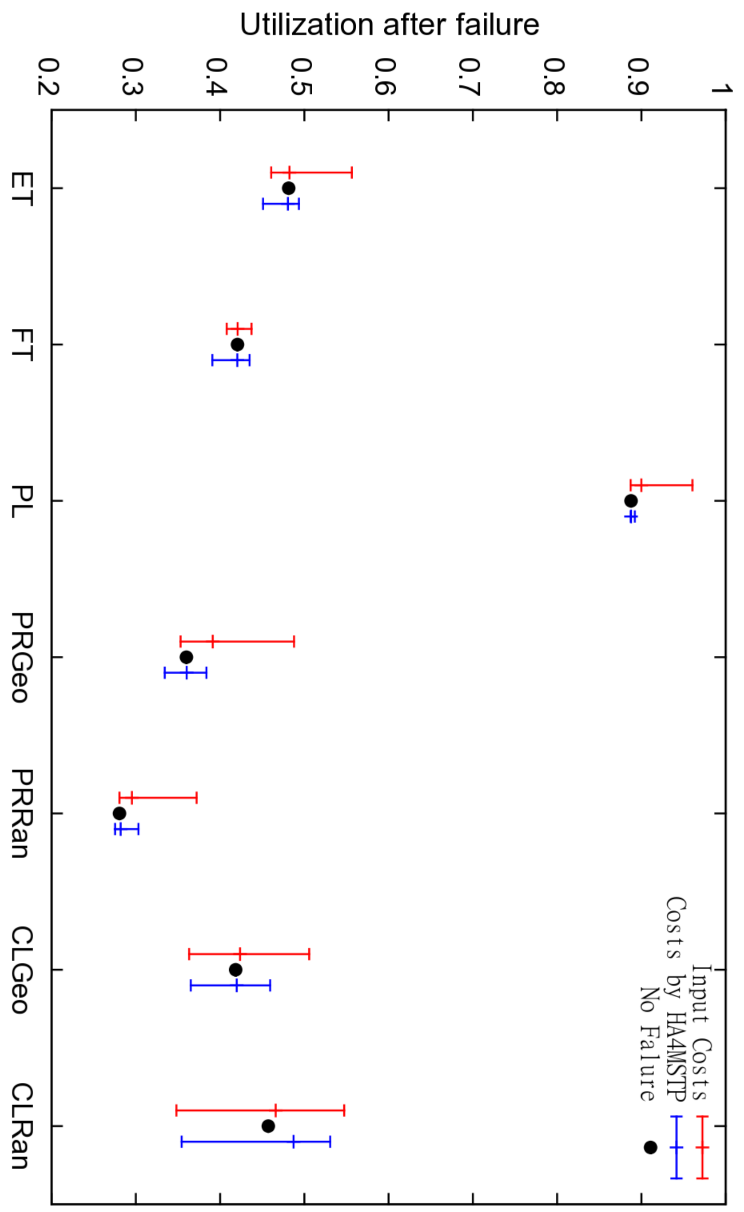


Figure 5.3: Results with MSTP default costs

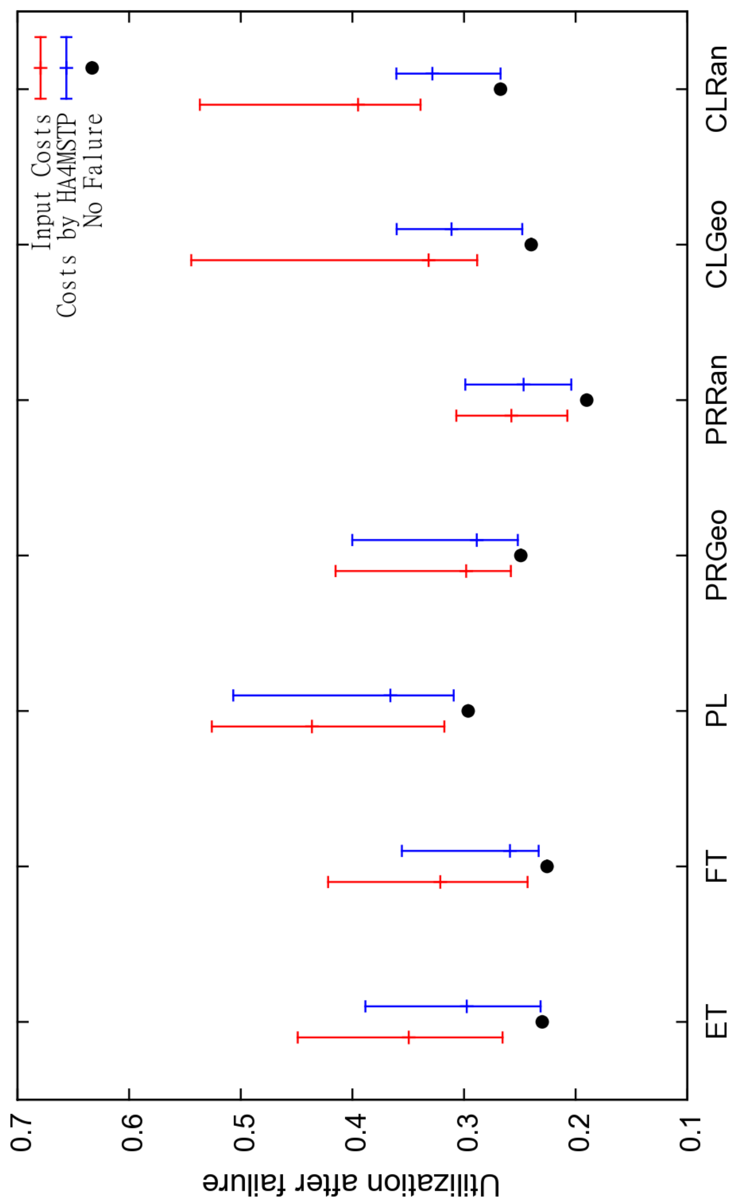


Figure 5.4: Results with LSA4MSTP optimized costs

Table 5.3: Results with default input costs (HA stands for HA4MSTP)

Topo.	$avgChg$	$maxChg$	$minChg$	$U_{nofails}$	U_{fails}		Time Exec. (min)	
	802.1s	802.1s	802.1s		802.1s	HA	802.1s	HA
Exp. Tree	2.28	18	1	0.4814	0.5564	0.4935	2.08	0.73
Fat Tree	1.65	23	1	0.4207	0.4372	0.4350	5.78	2.06
PortLand	2.35	42	1	0.8876	0.9606	0.8922	3.80	1.91
Private Geo.	2.61	17	1	0.3599	0.4878	0.3837	2.00	1.09
Private Ran.	3.34	20	1	0.2805	0.3721	0.3031	2.29	1.52
Cloud Geo.	2.24	31	1	0.4185	0.5058	0.4595	10.65	9.30
Cloud Ran.	2.38	31	1	0.4572	0.5472	0.5306	9.57	8.97

Table 5.4: Results with LSA4MSTP optimized costs

Topo.	$avgChg$	$maxChg$	$minChg$	$U_{nofails}$	U_{fails}		Time Exec.(min)	
	802.1s	802.1s	802.1s		802.1s	HA	802.1s	HA
Exp. Tree	3.51	26	1	0.2299	0.4489	0.3884	3.26	0.69
Fat Tree	10.96	43	1	0.2256	0.4217	0.3557	6.63	2.32
PortLand	5.82	49	1	0.2963	0.5259	0.5068	8.14	2.13
Private Geo.	4.97	29	1	0.2492	0.4150	0.4001	3.18	1.03
Private Ran.	4.66	29	1	0.1901	0.3069	0.2989	2.83	1.55
Cloud Geo.	9.83	78	1	0.2398	0.5443	0.3604	9.93	9.11
Cloud Ran.	9.73	130	1	0.2673	0.5366	0.3606	9.40	9.00

6 Multi-objective Traffic Engineering

In Chapter 5, we approached a multi-objective TE by solving a minimization problem with two objectives. However, the minimization of service disruption is considered as a hard constraint to be guaranteed. This is a special case because it is often impossible to reach the optimum of an objective in TE problems in large DCNs. In this chapter, we introduce a novel design of multi-objective algorithms for solving the TE problem in DCNs. We show how the heuristics of two single objective LSs can be included into the search process of a bi-objective LS. We also describe how we represent, update and evaluate the outcomes of the bi-objective algorithm. Different assessment metrics for multi-objective algorithms are adapted for evaluating the overall performance of our LS approaches.

6.1 Introduction

Solving the TE problem in large Ethernet networks can be defined as the optimization of the network quality of service (QoS) based on many evaluation metrics, such as link utilization, link delay, sum load, energy saving, number of used links, network convergence, etc . . . It is impossible to optimize all these metrics at the same time because an improvement in one metric can degrade other metrics. In practice, network operators would like to achieve the best network performance based on a compromise among a given set of metrics. Multi-objective optimization [YST85] is recommended for solving this kind of TE problem, where each objective is related to one metric. Solving the single objective problem is already very complex, as the problem is NP-hard. Thus, we focus on approximate methods based on local search.

In this chapter, we extend the TE problem defined in Chapter 4 but with two bi-objective optimization problems. The first one is the minimization of the maximal link utilization and the sum load. The second is the minimization of the maximal link utilization and the number of used links. The maximal link utilization appears in both problems because it is the essential metric for assessing the QoS of a TE technique. Our contribution is mainly focused on the design, development, analysis, and evaluation of multi-objective methods for the TE problem in data centers.

Related Work

Relating to the multi-objective TE, [HZC06] took into account link load balancing and admission fairness. The framework proposed in [LYD$^+$03] considered both network throughput and delay. Recently, [dSS07] aimed at minimizing the n worst link loads (with n ranging up to the total number of network links) and the average link load in the network. However, the final output of these approaches is a unique solution where the second metric is only considered as a soft constraint or a lexicographical objective. In practice, the outcome of multi-objective algorithms is often a set of non-dominated solutions called a Pareto front [YST85] (see Section 2.5.1 in Chapter 2). In this chapter, we propose two bi-objective algorithms that take into account the minimization of the maximal link utilization, sum load and number of used links. Our bi-objective algorithms aim to provide good Pareto fronts instead of a single solution as in the state-of-the-art approaches.

It is difficult to define a reliable and convincing method to validate a multi-objective algorithm that deals with an NP-hard problem without the reference of a global Pareto front (optimal solution in case of multi-objective optimization). Many assessment metrics for evaluating the quality of a Pareto front have been presented, such as Purity and Spread

Metric in [CMVV11], Hypervolume in [ZT98], and Binary ϵ-indicator in [ZTL$^+$03]. The drawback of these metrics is that each of them is not strong enough to evaluate the overall performance of multi-objective algorithms for TE in DCNs. For instance, the Purity and Binary ϵ-indicator metrics cannot guarantee the quality of a considered algorithm if its Pareto front contains few solutions. The Hypervolume and Spread Metric need a credible method for finding good reference points. In this work, we introduce a new strong evalution method using five different assessment metrics (described in Section 6.4.1) for validating the Pareto front given by bi-objective algorithms.

6.2 Problem Statement

As in the previous chapters, we model this TE problem as a graph problem with:

- Undirected graph $G = (N, E)$ where N is the set of switches and E is the set of links.
- BW - the matrix of link bandwidths.
- $V = \{V_1, V_2, \dots, V_k\}$ - the set of k given VLANs in the network, with $V_r = (N_r, E_r)(N_r \subseteq N,\ E_r = \{(i, j) \in E | i, j \in V_r\},\ 1 \leq r \leq k)$.
- $TD = \{TD_1, TD_2, \dots, TD_k\}$ - the set of k traffic demand matrices where $TD_r[i, j]$ represents the traffic that switch i sends to switch j in VLAN V_r ($1 \leq r \leq k$; $i, j \in V_r$).
- $W = \{W_1, W_2, \dots, W_k\}$ - the set of link cost matrices for the k given VLANs.
- $MSP(G, V, W)$ - the set of k spanning trees $SP_1(G, V_1, W_1)$, $SP_2(G, V_2, W_2)$,..., $SP_k(G, V_k, W_k)$ obtained with the Multiple Spanning Tree Protocol [Soc02] on graph G, with set of VLANs V and set of link cost matrices W.

Obviously, the only set that can be configured to change the solution (set of k spanning trees) is W. The link load L, the link utilization

U, and the maximal link utilization U_{max} are defined as in the previous chapters. We denote $SumL$ the sum of all link loads in the network: $SumL = \sum L[i, j]$ $(i, j \in N)$.

Let $\#Links$ denote the number of used links in the network. A link (i, j) is considered to be a used link if $L[i, j] + L[j, i] > 0$.

In this work, we perform three measures U_{max}, $SumL$ and $\#Links$ on each solution. Among these measures, the maximal link utilization value U_{max} is the most important for evaluating a TE solution. The ultimate objective is to avoid congestion in the network (keep the value of U_{max} under 100%). The second measure of the sum of all link loads $SumL$ is an important criterion for assessing the total traffic that flows on each link of the network and also to assess the network delay in some sense. The number of used links $\#Links$ is very significant in terms of network backup in case of link failures and energy efficiency in data centers [CSB+08]. It is impossible to obtain an optimal solution which minimizes U_{max}, $SumL$, and $\#Links$ at the same time. In our experiments, an improvement in $SumL$ or $\#Links$ often leads to a degradation of U_{max}. In addition, taking simultaneously three objective functions into account for a TE problem on large data centers is a very complex task. For this reason, we divide these three objectives into two pairs of objectives: (U_{max} , $SumL$) and (U_{max} , $\#Links$).

The formulations of these two bi-objective optimizations are the following:

PROBLEM 4
Input: Graph $G = (N, E)$, set of k VLANs V, bandwidth matrix BW, set of k traffic demand matrices TD
Output:

- Bi-objective 1 (Minimization of U_{max} and $SumL$): A Pareto front $PF_{(U_{max},SumL)}$, each solution in $PF_{(U_{max},SumL)}$ is a set of k spanning trees minimizing the pair of objectives U_{max} and $SumL$.

- Bi-objective 2 (Minimization of U_{max} and $\#Links$): A Pareto front $PF_{(U_{max},\#Links)}$, each solution in $PF_{(U_{max},\#Links)}$ is a set of k spanning trees minimizing the pair of objectives U_{max} and $\#Links$.

- Given a set of k spanning trees SP_i, determine a set of k cost matrices W_i $(1 \leq i \leq k)$ such that $MSP(W_i)$ generates the spanning tree SP_i (straightforward by applying the technique proposed in Section 3.4 in Chapter 3 for each VLAN in V).

We describe in Section 6.4 the assessment methods for evaluating the quality of the Pareto front obtained by our proposed LS algorithms (in Section 6.3) for solving these two bi-objective problems.

6.3 Proposed Algorithms

In this work, we present five LS algorithms dealing with the TE problems described in Section 6.2. We first implement three LS algorithms that take into account each of the three objectives to be minimized (U_{max}, $SumL$ and $\#Links$). To obtain a bi-objective Pareto front with these algorithms, we add an observation to store the non-dominated solutions in their search process. Next, two bi-objective algorithms are proposed based on the three single objective methods. We will refer to the three single-objective minimization algorithms as $Min_{U_{max}}$, Min_{SumL}, and $Min_{\#Links}$; the two bi-objective algorithms will be referred to as $Min_{(U_{max},SumL)}$ and $Min_{(U_{max},\#Links)}$.

6.3.1 Single-objective Approaches

Minimization of U_{max}
Our LSA4MSTP in Chapter 4 shows good perfomance with the minimization of U_{max} when it could reduce more than 50% the value of U_{max} compared to the solution given by MSTP 802.1s for large DCNs in our experiments.

Minimization of $SumL$
The Optimum Communication Spanning Tree Problem (OCT) [Hu74] can be considered as a sub problem of this minimization of sum load in network with one VLAN containing all its switches. OCT is a very challenging combinatorial optimization problem and is proven to be NP-hard even with a unit traffic demand matrix ($TD[i,j] = 1, \forall i \neq j$).

Our Min_{SumL} LS is inspired by the $Min_{U_{max}}$ algorithm with the same definitions of the search space and speeding up techniques.

The key design of each LS algorithm is the definition of the neighborhood. Min_{SumL} also relies on an edge exchange in each search iteration to move from one solution to another. To find a heuristic for this LS, we represent $SumL$ by another formula:

$$SumL = \sum_{i,j \in N} L[i,j] = \sum_{\substack{r \in [1..k] \\ i,j \in V_r}} TD_r[i,j] * PathLength_r[i,j],$$

where k is the number of VLANs, and $PathLength_r[i,j]$ is the number of links on the path from i to j in the VLAN V_r.

With this new definition, we can state that the overall sum load $SumL$ can be changed by modifying the most contributing pair (s,t), in which the sum load over path $TD_r[s,t] * PathLength_r[s,t]$ is large. In each search iteration, we first choose a pair (s,t) from x pairs having the largest sum load over path $TD_r[s,t] * PathLength_r[s,t]$. Next, we apply a heuristic for selecting the edge to be removed on the path from s to t and the edge to be added for minimizing the value of $SumL$. We also use time as termination criterion, and a tabu list for this LS.

The speeding up techniques in $Min_{U_{max}}$ are reused to accelerate the search. The main computation in this Min_{SumL} is the computation and comparing $SumL$ in each search iteration. Obviously, the traffic demand matrices are unchangeable, so the computation of path length is the most important task. Fortunately, we can maintain incrementally the path length between each pair of switches, relying on the "magic cycle" described in Section 3.3.5 in Chapter 3.

When the traffic demand matrices are nearly uniform, the minimization of $SumL$ is equivalent to the minimization of the average path length between each pair of switches in the network. For this reason, the balance spanning tree construction of STP 802.1d [Soc98] with the root in the center is also a very efficient method to reduce the overall sum load. Thus, with the initial solution obtained with 802.1s, Min_{SumL} reduces by less than 1% the value of $SumL$, but compared to a random initial solution, this percentage of reduction is about 12% in our experiments.

Minimization of $\#Links$

For this $Min_{\#Links}$, we also iterate a move to a neighbor solution by performing an edge exchange in each search iteration. Our heuristic for defining the neighborhood for $Min_{\#Links}$ is based on the following observation:
The number of used links by a set of k spanning trees can be reduced only if in a spanning tree SP_i, we replace an edge used only by SP_i by an edge used by another spanning tree SP_j.
Let the $numVLAN$ of an edge (s, t) be the number of VLANs that have (s, t) in their spanning trees. From the initial solution obtained with 802.1s, a set of removable edge candidates RM is created from the edges with the first x minimal values of $numVLAN$. In each search iteration, we randomly select an edge (s_0, t_0) in RM to be replaced by an edge (s_I, t_I). We accept only a replacement that reduces or maintains the current $\#Links$. A random selection of (s_0, t_0) is performed after a given number of non-improving iterations.

In our experiments, the solutions obtained by $Min_{\#Links}$ can reduce by about 16$\%$ the value of $\#Links$ compared to the solution given by 802.1s.

Solution quality
In Table 6.1, we measure the performance of each single objective algorithm compared to the default configuration of the MSTP 802.1s standard. The data set for these experiments is described in Section 6.4.2. In our tests, a triple measure of $(U_{max}, SumL, \#Links)$ is performed on the best solution given by the three single objective LS. We assume a value of 100$\%$ for the minimum of U_{max}, $SumL$, and $\#Links$. If we only consider the visible improvement in each single objective compared to the MSTP 802.1s standard, then $Min_{U_{max}}$ seems to be the best algorithm and Min_{SumL} is possibly the worst one. However, in order to reduce by 233.26$\%$ the value of U_{max} compared to 802.1s, $Min_{U_{max}}$ uses 125.6$\%$ more links than $Min_{\#Links}$ (about 500 more links in a data center with 2000 links) and increases $SumL$ by 120.25$\%$ more than Min_{SumL}. It is even worse for $Min_{\#Links}$ where its best solution increases U_{max} by 346.32$\%$ over that given by $Min_{U_{max}}$, and $SumL$ by 131.69$\%$ compared to that obtained by Min_{SumL} in order to reduce 118.51$\%$ $\#Links$ from the initial solution given by 802.1s. Hence, the overall solution quality of an algorithm cannot be evaluated by its best

Table 6.1: Single objective algorithm overall performance

%\Alg.	802.1s	$Min_{U_{max}}$	Min_{SumL}	$Min_{\#Links}$
U_{max}	233.26	**100**	221.69	346.32
$SumL$	100.57	120.25	**100**	131.69
$\#Links$	118.51	125.6	119.29	**100**

solution for a single objective.

6.3.2 Bi-objective Optimization

Of our three single objective LS, we found that $Min_{U_{max}}$ converges much faster than Min_{SumL} or $Min_{\#Links}$. Fig. 6.1 depicts the improvement of U_{max} over time of $Min_{U_{max}}$ for a data center with 564 switches and 16 VLANs with a time limit of 15 minutes. $Min_{U_{max}}$ reduces U_{max} by 52.2% compared to that of 802.1s (from 0.69 to 0.33) after only 10s. With the same test, the improvement within the first 10 s of Min_{SumL} is 3.6% and for $Min_{\#Links}$, 1.77%.

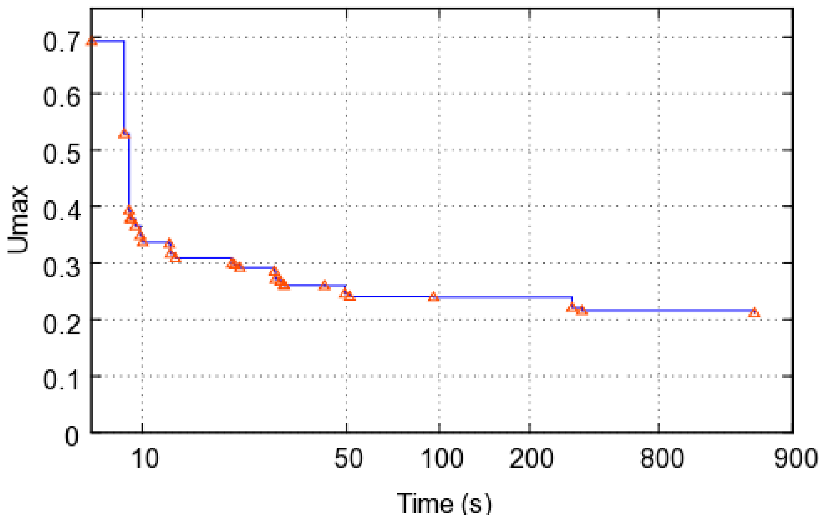


Figure 6.1: Improvement of U_{max} over execution time

When we look at the number of improvements that each algorithm makes within the first 10 s, $Min_{U_{max}}$ improved its objective value only 6 times while this number is 18 for Min_{SumL} and 13 for $Min_{\#Links}$. This convergence observation motivates our algorithmic approach to the two bi-objective optimizations: $Min_{(U_{max},SumL)}$ and $Min_{(U_{max},\#Links)}$. Here, we do not consider the bi-objective $(SumL, \#Links)$ because U_{max} is the essential metric in a TE problem and we cannot guarantee for the network congestion avoidance with only an optimization of $(SumL, \#Links)$.

Minimization of U_{max} and $SumL$

Algorithm 9 describes how the heuristics of two single objective LSs are included in the search process of the bi-objective LS. In this case, we combine $Min_{U_{max}}$ with Min_{SumL} to define the $Min_{(U_{max},SumL)}$ algorithm. In $Min_{(U_{max},SumL)}$, $SumL$ is considered as Objective 1 to be regularly minimized (Line 8). The $edgeExchangeHeuristicObj1(MSP,$-$G,V)$ returns a neighboring solution of the current solution MSP that minimizes the $SumL$ value by applying the edge exchange heuristic of Min_{SumL}. With its fast convergence, the edge exchange heuristics of $Min_{U_{max}}$ are periodically applied in the method $edgeExchangeHeuristicObj2(MSP, G, V)$ (Line 10) after each $\#stepObj1$ iterations to guide the search by minimizing the value of Objective 2 – U_{max}. The $\#stepObj1$ in Line 7 is an important parameter for configuring to balance the convergence of U_{max} and $SumL$.

The Pareto front are updated with the non-dominated solutions found during the search process (Line 19). We diversify the search with a random edge replacement after a given number of non-improving search iterations (Lines 13-16).

Minimization of U_{max} and $\#Links$

The $Min_{(U_{max},\#Links)}$ LS is also defined by the pseudo code in Algorithm 9. Here, $\#Links$ becomes Objective 1 and we apply the $Min_{U_{max}}$ edge exchange heuristic to guide the search after every $\#stepObj1$. A random move is also applied to diversify the search when no new non-dominated solutions are found in a given number of successive search iterations.

Both $Min_{(U_{max},SumL)}$ and $Min_{(U_{max},\#Links)}$ share the same design

Algorithm 9: Bi-objective Algorithms

```
$MSP$ = getInitialSolution($G, V$); /* by MSTP 802.1s */
$PF = \{MSP\}$; /* Insert MSP to Pareto front */
$\#iterations = 0$;
$\#nonImprovements = 0$;
while $time\_exec < time\_windows$ do
    $\#iterations = \#iterations + 1$;
    if $\#iterations \ \% \ \#stepObj1 \neq 0$ then
        $MSP = edgeExchangeHeuristicObj1(MSP, G, V)$ ;
    else
        $MSP = edgeExchangeHeuristicObj2(MSP, G, V)$ ;
    end
    if MSP is dominated by PF then
        $\#nonImprovements = \#nonImprovements + 1$;
        if $\#nonImprovements > \#acceptIterations$ then
            $MSP = edgeExchangeRandom(MSP, G, V)$;
        end
    else
        $\#nonImprovements = 0$;
        $PF = updateParetoFront(PF, MSP)$;
    end
end
```

described in Algorithm 9 but they are very different in their exploration of the huge search space: they apply different definitions of neighbors. The configuration of the two parameters $\#acceptIterations$ and $\#stepObj1$ is also different in each algorithm.

Finding Pareto front with single objective algorithms

In the assessment of our bi-objective optimization algorithms, we would like to compare their performance with single objective algorithms. Such a comparison is possible by extending our single objective algorithms to compute a Pareto front. The idea is just to observe a second objective and compute a Pareto front. In Algorithm 10, we present how a Pareto front PF can be obtained with a single objective algorithm. Here, an improvement in the unique objective is normally stored (Line 8-10) but a dominance check is also added in each search iteration to update the Pareto front PF with the second objective (Line 12, 13). By adding this observation, we have four more Pareto fronts in each experiment: two PFs of (U_{max}, $SumL$) obtained with $Min_{U_{max}}$ and Min_{SumL}; and two PFs of (U_{max}, $\#Links$) obtained with $Min_{U_{max}}$ and $Min_{\#Links}$.

Algorithm 10: Finding a Pareto front with a single objective algorithm

```
1  $MSP$ = getInitialSolution($G, V$);       /* by MSTP 802.1s standard */
2  $PF = \{MSP\}$; /* insert MSP to Pareto front */
3  $MSP^* = MSP$;   /* MSP* best solution wrt obj1 */
4  $bestObj1Value = getObj1Value(MSP)$;
5  while $time\_exec < time\_windows$ do
6      $MSP = edgeExchangeHeuristicObj1(MSP, G, V)$ ;
7      $obj1Value = getObj1Value(MSP)$;
8      if $bestObj1Value > obj1Value$ then        /* stock improvement */
9          bestObj1Value=obj1Value;
10         $MSP^* = MSP$;
11     end
12     if MSP is not dominated by PF then        /* check dominance to update PF */
13         $PF = updateParetoFront(PF, MSP)$;
14     end
15 end
```

In the next section, we will use different assessment methods for evaluating our proposed algorithms on different network topologies.

6.4 Performance Comparisons

This section is organized as follows. We first describe the evaluation methods for multi-objective approaches in Section 6.4.1. Next, the network topologies and traffic demand matrices for our tests are presented in Section 6.4.2 Last, our experimental results are discussed in Section 6.4.3.

6.4.1 Assessment Metrics for Multi-objective Algorithms

Many measurements can be made for evaluating the outcomes of multi-objective algorithms. In this chapter, we consider five metrics to use in comparing the performances of our local search approaches: Purity [CMVV11], Spread Metric [CMVV11], Hypervolume [ZT98], Hypervolume-Star, and Binary ϵ-indicator [ZTL$^+$03]. These five metrics are very useful for assessing the quality of an approximation set or a Pareto front. In this work, we denote by PF_{obt} the Pareto front produced by the considered algorithm and PF_{ref} the reference (or best-known) Pareto front. PF_{ref} is the best Pareto front, combining the Pareto fronts PF_{obt} of all the considered algorithms. In this section, we assume without loss of generality that all the multi-objective algorithms aim at solving the minimization problem with two objectives.

Purity

The purity metric [CMVV11] measures the percentage of solutions of PF_{obt} that are both in PF_{obt} and in PF_{ref}:

$$Purity = \frac{|PF_{\text{obt}} \bigcap PF_{\text{ref}}|}{|PF_{\text{obt}}|},$$

where $|PF_{\text{obt}}|$ is the number of solutions of PF_{obt} and $|PF_{\text{obt}} \bigcap PF_{\text{ref}}|$ is the number of solutions contained both in PF_{obt} and in PF_{ref}.

Obviously, $Purity \in [0, 1]$, $Purity$ equals 0 when PF_{obt} and PF_{ref} do not share any common solution (all the solutions of PF_{obt} are dominated by some solutions of PF_{ref}), $Purity$ equals 1 if PF_{obt} is completely included in PF_{ref} (no other algorithm was able to compute a solution dominating a solution of PF_{obt}), and a large $Purity$ means a good multi-objective algorithm. This metric is significant when both PF_{obt} and PF_{ref} have many solutions. However, if $PF_{\text{obt}} = \{s^*\}$ and $s^* \in PF_{\text{ref}}$, then $Purity = 1$. In this case, we have no guarantee of the quality of the considered algorithm.

Spread Metric

The spread metric (Γ) [CMVV11] measures the maximum size of the "holes" of a Pareto front. Suppose given $PF_{\text{obt}} = \{s_1, s_2, .., s_n\}$ with $s_i = (f_{i,1}, f_{i,2})$. We assume that the objective values are normalized and are sorted in increasing order for objective 1 and in decreasing order for objective 2. We consider two extreme points s_0 and s_{n+1} of PF_{obt} such that

$$f_{0,1} \leq f_{1,1} \text{ and } f_{0,2} \geq f_{1,2},$$
$$f_{n+1,1} \geq f_{n,1} \text{ and } f_{n+1,2} \leq f_{n,2}$$

The δ metric is defined by

$$\delta_{i,j} = |f_{i,j} - f_{i-1,j}|, i \in [1..n+1], j \in [1..2]$$

where $f_{i,j}$ is the value of objective function j of solution $s_i \in PF_{\text{obt}} \bigcup \{s_0, s_{n+1}\}$. The spread metric Γ is defined by

$$\Gamma = max\{i \in [1..n+1], j \in [1..2]; \delta_{i,j}\}$$

Fig. 6.2b depicts an example of a Γ measurement with $m = 3$, Γ takes the largest value $\delta_{4,1}$.
By this definition, PF_{obt} is good if its maximal hole Γ is small. However, we found that PF_{obt1} could have a smaller Γ than PF_{obt2} even if all the solutions of PF_{obt1} are dominated by some solutions of PF_{obt2}. Thus, we define a new spread metric Γ^*, measuring the spread (or coverage) of PF_{obt} over a global Pareto front PF_{ref}.
We use two extreme points s_0 and s_{n+1} of PF_{ref} to compute both $\Gamma(PF_{\text{obt}} \bigcap PF_{\text{ref}})$ and $\Gamma(PF_{\text{ref}})$. The Γ^* metric is defined by

$$\Gamma^* = \frac{\Gamma(PF_{\text{ref}})}{\Gamma(PF_{\text{obt}} \bigcap PF_{\text{ref}})}$$

If $PF_{\text{obt}} \bigcap PF_{\text{ref}} = \emptyset$, then Γ^* is undefined. So we set $\Gamma^* = 0$ as the worst case when PF_{obt} does not contain any solution in PF_{ref}. With this assumption, $\Gamma^* \in [0, 1]$. Γ^* equals 1 (ideal) if $PF_{\text{obt}} = PF_{\text{ref}}$.

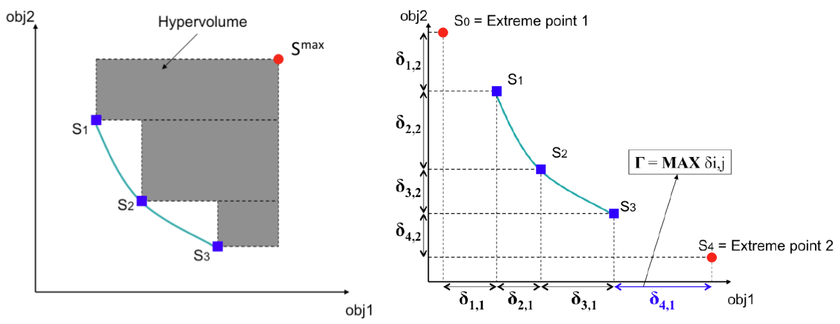
Finding Extreme Points for Γ^*

In [CMVV11], the authors proposed an experimental method for computing the two extreme points for Γ by selecting the pair of objective

values corresponding to the highest pairwise distance. This selection must be performed on the non-dominated solution set over all runs in order to obtain the two final extreme points. This method is expensive and incompatible with the computation of our new metric Γ^*.
For the computation of Γ^*, we found that the two extreme points cannot belong to PF_{obt} or PF_{ref}. Let $s_0 = (f_{0,1}, f_{0,2})$ and $s_{n+1} = (f_{n+1,1}, f_{n+1,2})$ be two extreme points for $PF_{\text{ref}} = \{s_1, s_2, .., s_n\}$ with $s_i = (f_{i,1}, f_{i,2})$ and $n > 1$. If $n = 1$, the extreme points do not need to be defined, $\Gamma^* = 0$ if $PF_{\text{obt}} \bigcap PF_{\text{ref}} = \emptyset$ and $\Gamma^* = 1$ if $PF_{\text{obt}} \bigcap PF_{\text{ref}} = PF_{\text{ref}}$. We assume that the objective values are normalized and are sorted in increasing order for objective 1 and in decreasing order for objective 2. We put $Dist_1 = \frac{f_{n,1} - f_{1,1}}{n-1}$ and $Dist_2 = \frac{f_{1,2} - f_{n,2}}{n-1}$. The extreme point values $f_{0,1}, f_{0,2}, f_{n+1,1}$ and $f_{n+1,2}$ are defined by

$$\begin{gathered} f_{1,1} - Dist_1 \leq f_{0,1} \leq f_{1,1} \text{ and } f_{1,2} < f_{0,2} \leq f_{1,2} + Dist_2, \\ f_{n,1} < f_{n+1,1} \leq f_{n,1} + Dist_1 \text{ and } f_{n,2} - Dist_2 \leq f_{n+1,2} \leq f_{n,2} \text{ (1)} \end{gathered}$$

Our definition of $f_{0,1}, f_{0,2}, f_{n+1,1}$ and $f_{n+1,2}$ in (1) ensures that the distance between s_0 - s_1 and s_n - s_{n+1} is always less than or equal Γ. In addition, this definition of extreme points is more compatible with Γ^* because it allows us to better represent the contribution of PF_{obt} to PF_{ref}. In this work, we fix $s_0 = (f_{1,1}, f_{1,2} + Dist_2)$ and $s_{n+1} = (f_{n,1} + Dist_1, f_{n,2})$.



(a) Hypervolume (b) Gamma Measurement

Figure 6.2: Measurement methods

Hypervolume

The hypervolume (HP) metric [ZT98] measures the "volume" of the dominated region defined by a Pareto front and a reference point s^{max}. Suppose given $PF_{\text{obt}} = \{s_1, s_2, .., s_n\}$ with $s_i = (f_{i,1}, f_{i,2})$. Let $s^{\text{max}} = (f_1^{max}, f_2^{max})$ be a reference point such that $f_1^{max} \geq max_{i\in[1..n]}\{f_{i,1}\}$ and $f_2^{max} \geq max_{i\in[1..n]}\{f_{i,2}\}$. HP is defined as the surface area of the union of all the rectangles defined by each of the solutions in PF_{obt} and the reference point s^{max}.
Fig. 6.2a presents an example of the hypervolume computation of a Pareto front obtained from a bi-objective algorithm. In this work, we normalize the values to [0,1] for each objective ($s^{\text{max}} = (1, 1)$). Thus, $HP \in [0, 1]$. With HP, the quality of a Pareto front is evaluated by the surface of the dominated region which it covers. A method with a large HP is considered better than a method with a smaller HP. HP is useful for comparing the performance of different algorithms solving a specific multi-objective problem. However, with two different reference points, a Pareto front PF_{obt1} could cover a bigger region than a Pareto front PF_{obt2} even when the solutions of PF_{obt1} are completely dominated by the solutions of PF_{obt2}. So, a unique reference point is required when comparing different Pareto fronts.

Finding the Reference Point s^{max}

In [ZT98], the authors simply used the point (0,0) in the context of a maximization problem as the reference point for computing the hypervolume HP of its Pareto front. However, if the considered Pareto fronts are too far from (0,0), the "volume" of HP consists of a large surface of the insignificant region that can make the HP metric inexact. We have the same problem with the minimization problem if the reference point is too far from the Pareto front considered.
For a Pareto front $PF_{\text{obt}} = \{s_1, s_2, .., s_n\}$ with $s_i = (f_{i,1}, f_{i,2})$, we define its upper bound point $s^* = (f_1^*, f_2^*)$ as

$$f_1^* = f_{n,1} + Dist_1,\ f_2^* = f_{1,2} + Dist_2, \text{ if } n > 1,$$
$$f_1^* = f_{1,1} + \tfrac{1}{2}.max(f_{1,1}, f_{1,2}),\ f_2^* = f_{1,2} + \tfrac{1}{2}.max(f_{1,1}, f_{1,2}), \text{ if } n = 1 \quad (2)$$

Given m Pareto fronts to be compared $PF_{\text{obt1}}, PF_{\text{obt2}}, ..., PF_{\text{obt}m}$, let $s_i^* = (f_{i,1}^*, f_{i,2}^*)$ be the upper bound point of $PF_{\text{obt}i}$. The unique reference point $s^{\max} = (f_1^{max}, f_2^{max})$ for computing the HP of these Pareto fronts is defined by

$$f_1^{max} = max_{i \in [1..m]}(f_{i,1}^*) \text{ and } f_2^{max} = max_{i \in [1..m]}(f_{i,2}^*)$$

Our definition of reference point for HP is based on the solution values of all the Pareto fronts considered. In (2), an additional distance is added to the maximal values of each objective ensuring that s_1 and s_n contribute to the HP of each Pareto front considered.

Hypervolume-Star

The hypervolume-star (HP^*) metric is a combination of the purity and the hypervolume metric. We introduce this new hypervolume metric to disable the dependence of the purity on the number of solutions. We define the HP^* metric of a Pareto front PF_{obt} over a global Pareto front PF_{ref}) by

$$HP^* = \frac{HP(PF_{\text{obt}} \bigcap PF_{\text{ref}})}{HP(PF_{\text{ref}})}$$

We have $HP^* \in [0, 1]$ because $PF_{\text{obt}} \bigcap PF_{\text{ref}} \subseteq PF_{\text{ref}}$. HP^* measures the contribution of PF_{obt} in the hypervolume PF_{ref}. HP^* equals 0 if $PF_{\text{obt}} \bigcap PF_{\text{ref}} = \emptyset$ (PF_{obt} and PF_{ref} do not share any common solution). HP^* equals 1 if $PF_{\text{obt}} \bigcap PF_{\text{ref}} = PF_{\text{ref}}$ (ideal when PF_{obt} is the global Pareto front PF_{ref}). In the computation of HP^*, one should use the same reference point $s^{\max}$ for computing $HP(PF_{\text{obt}} \bigcap PF_{\text{ref}})$ as for $HP(PF_{\text{ref}})$.

For computing HP*, we only consider the subset $PF_{\text{obt}} \bigcap PF_{\text{ref}}$ of PF_{ref}. Thus, the upper bound point s_{ref}^* of PF_{ref} (computed by formula (2)) is used as its reference point $s^{\max}$ ($s^{\max} = s_{ref}^*$).

Binary ϵ-indicator

For a bi-objective minimization problem, we define when a solution $s_1 = (f_{1,1}, f_{1,2})$ $\epsilon-$ dominates a solution $s_2 = (f_{2,1}, f_{2,2})$, denoted as $s_1 \succeq_\epsilon s_2$ as follows:

$$(s_1 \succeq_\epsilon s_2) \Leftrightarrow \forall i \in [1..2] : f_{i,1} \leq \epsilon.f_{i,2}$$

The ϵ-indicator $I(PF_{\text{obt}}, PF_{\text{ref}})$ is the minimum factor ϵ such that for any solution s_2 in PF_{ref} there exist at least one solution s_1 in PF_{obt} that is not worse by a factor ϵ in all objectives [ZTL+03].

$$I(PF_{\text{obt}}, PF_{\text{ref}}) = \inf_{\epsilon \in \mathbb{R}} \{\forall s_2 \in PF_{\text{ref}}, \exists s_1 \in PF_{\text{obt}} : s_1 \succeq_\epsilon s_2\}$$

We have $I(PF_{\text{obt}}, PF_{\text{ref}}) \in]0, 1]$. The ϵ-indicator $I(PF_{\text{ref}}, PF_{\text{obt}})$ compares the distance of PF_{obt} with PF_{ref}. With this metric, when comparing with PF_{ref}, the ideal value is $I(PF_{\text{ref}}, PF_{\text{obt}}) = 1$. This occurs when $PF_{\text{obt}} \subseteq PF_{\text{ref}}$: none of the solutions of PF_{obt} is dominated by a solution of PF_{ref}. All of the solutions of PF_{obt} are thus in the global Pareto front PF_{ref}. As to the purity, the limitation of this measure is that it does not consider the number of solutions of PF_{obt}. If $PF_{\text{obt}} = \{s^*\}$ and $s^* \in PF_{\text{ref}}$, then $I(PF_{\text{ref}}, PF_{\text{obt}}) = 1$ although one cannot thus assess the quality of PF_{obt}.

6.4.2 Data Set Composition

We use the same seven data center topologies as described in Chapter 5 for testing our algorithms (see Table 6.2 with two VLAN generation types Geographic (Geo.) and Random (Ran.), 16 VLANs and the near-uniform traffic demand matrices for every topology type in our experiments. These data sets are available online in [HDB12]. The time limit for running our algorithms was fixed to 30 minutes for all the network topologies considered.

Table 6.2: Data Set Composition

Size\Type	PR Geo\Ran	CL Geo\Ran	FT Ran	PL Ran.	ET Ran
Num. Nodes	242	564	320	313	200
Num. Links	549	2024	2048	602	398

6.4.3 Results and Evaluation

In this section, the Pareto fronts obtained with our five proposed LS algorithms in Section 6.3 are evaluated on the same data sets (Table 6.2). For each bi-objective problem, PF_{ref} is the Pareto front which is the union of all the solutions produced by all the algorithms considered. In each test, the Pareto fronts PF_{obt} obtained by $Min_{U_{max}}$, Min_{SumL}, and $Min_{(U_{max},SumL)}$ are compared in the bi-objective 1 problem: minimization of U_{max} and $SumL$. In the same manner, the Pareto fronts PF_{obt} obtained by $Min_{U_{max}}$, $Min_{\#Links}$, and $Min_{(U_{max},\#Links)}$ are evaluated in the bi-objective 2 problem: minimization of U_{max} and $\#Links$.

In this section, we assume without loss of generality that all the objective values are normalized to [0,1]. Thus, in each test, the extreme point 1 is $s_0(0,1)$, the extreme point 2 is $s_{n+1}(1,0)$ and the reference point is $s^{\text{max}}(1,1)$. Each Pareto front is assessed by the five metrics described in Section 6.1: Purity, Hypervolume (HP), Hypervolum-Star (HP*), Binary ϵ-indicator (I), and Spread Metric (Γ^*). The values of each metric are in [0,1], and the value 0 is the worst and 1 is the ideal.

Our detailed results for the seven network topologies in Table 6.2 are presented in Tables C.1 to C.14 (see Appendix) with these five assessment metrics. Each table describes the evaluation for one bi-objective problem $(U_{max}, SumL)$ or $(U_{max}, \#Links)$ on a given test. Table 6.3 represents the overall performance for the bi-objective problem 1 $(U_{max}, SumL)$ by computing the average value of each metric from the seven tables for $(U_{max}, SumL)$. Table 6.4 provides the same computations for $(U_{max}, \#Links)$. In our experiments, we also report the number of solutions in each Pareto front ($\#Sol$) as a reference assessment metric for better understanding the outcomes given by the five principal metrics.

Bi-objective problem 1: Minimization of U_{max} and $SumL$

In Table 6.3, we can see that none of the five metrics of $Min_{U_{max}}$ is the best, although its PF_{obt} has more than 24 solutions. Min_{SumL} has the best value for the metrics $Purity$ and I. However, these metrics are significant only when PF_{obt} has many solutions but PF_{obt} of Min_{SumL} has on average only 1.5 solutions. $Min_{(U_{max},SumL)}$ has the best value for three metrics: HP, HP*, and Γ^*, and its values for $Purity$ and I are very close to those of Min_{SumL}, with more than 12 solutions in its PF_{obt}. As expected, $Min_{(U_{max},SumL)}$ is the best method for solving this bi-objective problem.

Fig. 6.3 depicts the assessment metrics for $(U_{max}, SumL)$ on an expanded tree topology. None of the five metrics of $Min_{U_{max}}$ is the

Table 6.3: Performance Comparison U_{max}, $SumL$

PF \ Metric	#Sol	Purity	HP	HP*	I	Γ^*
$Min_{U_{max}}$	24.3	0.2884	0.6788	0.3971	0.8830	0.4986
$Min_{(U_{max},SumL)}$	12.7	0.8946	**0.8683**	**0.9158**	0.9846	**0.6640**
Min_{SumL}	1.5	**0.8961**	0.2075	0.1650	**0.9884**	0.3883

best because most of its solutions in PF_{ref} have a large value for $SumL$, and its PF_{obt} is mainly contained in the dominated region ($Purity = 0.2884$). Thus, the surface of the dominated region covered by $Min_{U_{max}}$ measured with HP* (in Fig. 6.3b) is much smaller than this surface measured with HP (in Fig. 6.3a).

In Fig. 6.3, we can state that Min_{SumL} has the best value for the metrics Purity and I, because its PF_{obt} has only one solution, which is contained in PF_{ref}. The PF_{obt} of $Min_{(U_{max},SumL)}$ is completely included in the center of PF_{ref} and it dominates many solutions of $Min_{U_{max}}$. Its $Purity$, 0.8946, is very large compared to this value for $Min_{U_{max}}$. However, $Min_{U_{max}}$ and Min_{SumL} are clearly better than $Min_{(U_{max},SumL)}$ from the single objective performance.

The $Min_{(U_{max},SumL)}$ LS provides a good Pareto front by ensuring a small value of $SumL$ with a significant decrease of U_{max} compared to the solution obtained by 802.1s (see Table 6.3). Futhermore, the PF_{ref} obtained by these three LSs offers a wide range of solutions for solving this TE problem.

Bi-objective problem 2: Minimization of U_{max} and $\#Links$

As expected, $Min_{(U_{max},\#Links)}$ is the best method when it has the best value for all the five assessment metrics (see Table 6.4). $Min_{(U_{max},\#Links)}$ has a $Purity$ of 0.9964 much larger than the $Purity$ of $Min_{U_{max}}$ or $Min_{\#Links}$.

Fig. 6.4 depicts a measurement for $(U_{max}, \#Links)$ on a cloud data center topology with geographic VLAN generation. In this test, all the non-dominated solutions given by $Min_{(U_{max},\#Links)}$ can be found at the center of PF_{ref}. As shown in Fig. 6.4b, the hypervolume covered by $Min_{(U_{max},\#Links)}$ is the largest. We can state that by keeping a small value of U_{max} and $\#Links$ for all its solutions, $Min_{(U_{max},\#Links)}$ provides good choices for network operators. In Fig 6.4, $Min_{(U_{max},\#Links)}$ can reduce the value of U_{max} from 0.7 to 0.3 using only 720 links while $Min_{U_{max}}$ needs 750 links to approximate U_{max}. In addition, we can easily choose a solution in PF_{obt} of $Min_{(U_{max},\#Links)}$ with a reasonable U_{max} (less than 0.5) which uses a small number of links in the network.

Table 6.4: Performance Comparison U_{max}, $\#Links$

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	**Γ***
$Min_{U_{max}}$	18.3	0.6611	0.2667	0.3515	0.8729	0.5812
$Min_{(U_{max},\#Links)}$	11.8	**0.9964**	**0.6972**	**0.9255**	**0.9992**	**0.7602**
$Min_{\#Links}$	7.0	0.3412	0.3090	0.1356	0.8745	0.5279

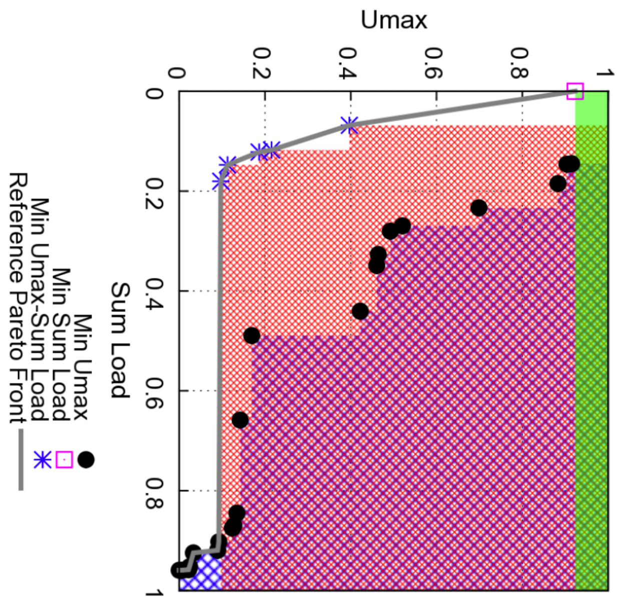
In this test, compared to the solution obtained with 802.1s MSTP standard, $Min_{\#Links}$ can reduce about 80 links while $Min_{U_{max}}$ markedly reduces the value of U_{max}: from 0.7 to about 0.2. Especially, the PF_{ref} is composed of the important contributions of all the PF_{obt} obtained with these three LSs.

6.5 Conclusion

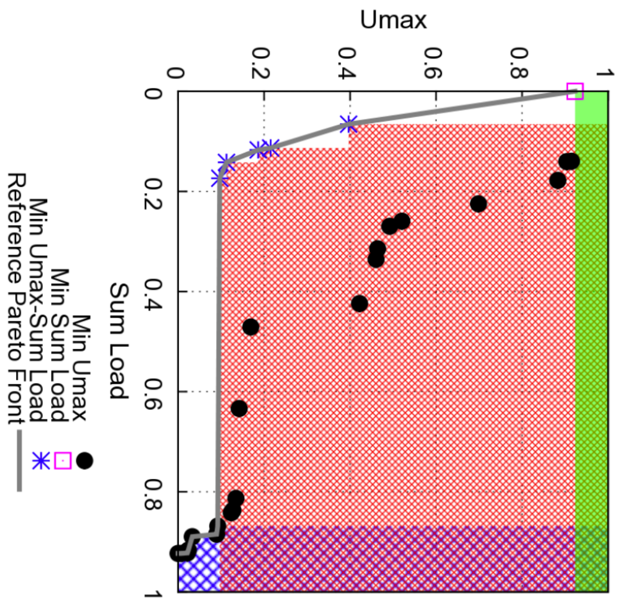
In this chapter, we turned to the multi-objective TE problem in DCNs. We proposed a new design of multi-objective algorithms, based on LS. Our algorithms show good performance for large data centers by integrating the heuristics of each single objective into the search process.

There are few existing assessment methods for evaluating the performance of multi-objective TE techniques in DCNs. The state of the art approaches often evaluate their outcomes based on each single objective observation. In this work, we introduced a strong assessment method for evaluating the multi-objective overall performance with three objectives: minimization of maximal link utilization U_{max}, sum load and number of used links. The non-dominated set of solutions (Pareto front) obtained by our bi-objective algorithms offers the network operators a wide range of good solutions for solving this TE problem in DCNs.

We think that it is interesting to adapt our design of multiple-objective algorithms for three or more objectives intead of two objectives. However, the real challenge is to produce a credible method for evaluating the algorithms with more than two objectives.

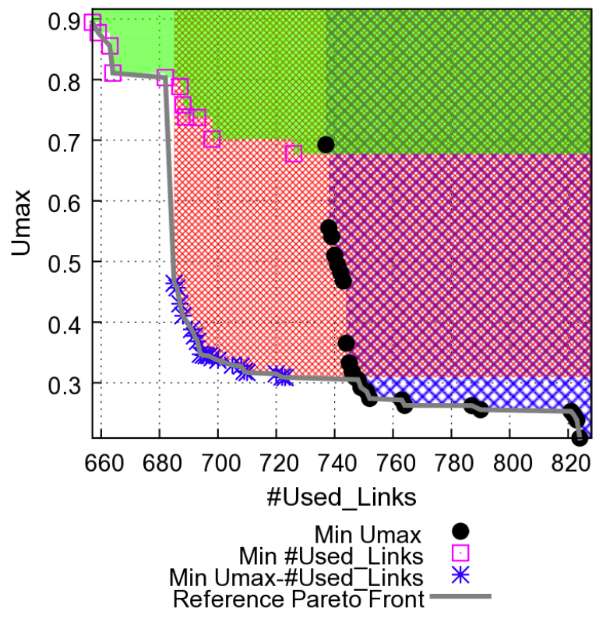


(a) Hypervolume HP

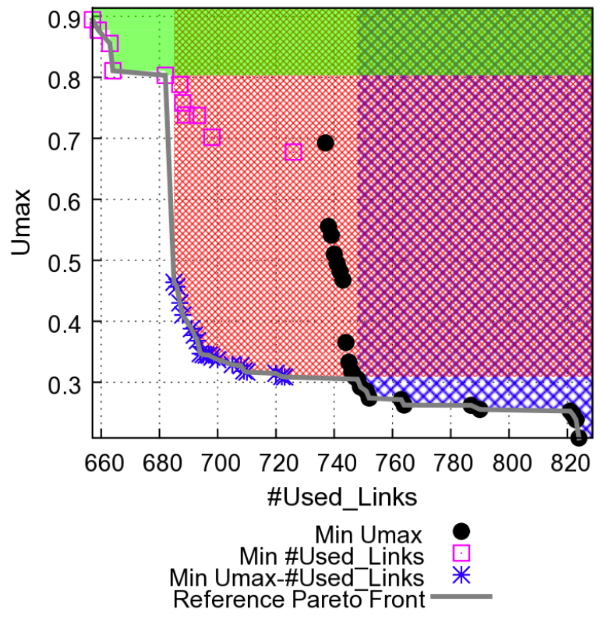


(b) Hypervolume-Star HP*

Figure 6.3: (U_{max}, $SumL$) measurement on Expanded Tree



(a) Hypervolume HP



(b) Hypervolume-Star HP*

Figure 6.4: $(U_{max}, SumL)$ measurement on Cloud Data Center

7 CONCLUSION

This thesis focussed on providing efficient TE techniques for improving the perfomance of large Ethernet networks, such as those found in data centers. Our approximate schemes are proposed for the TE problems considering different objectives in the networks that use STP 802.1d or MSTP 802.1s where network or each of its VLANs is reduced into a single spanning tree. We proved that our approximate approaches based on Local Search are very efficient for solving the NP-hard TE problems in the large networks. We summarize the main achievement of the thesis in Section 7.1. The limitations of our works and the possible directions for future work are presented in Section 7.2.

7.1 Results

In this thesis, four TE problems in large Ethernet networks have been addressed in each chapter by an increasing order of difficulty and of complexity.

As a first approach, chapter 3 coped with the minimization of the maximal link utilization U_{max} in Ethernet networks containing one spanning tree. Our TE technique based on local search aims to find the best (possible) spanning tree that minimizes congestion for a given traffic matrix. From this chapter, we learned that the choice of directly optimizing spanning trees instead of link weights reduces considerably the size of the search space. We proposed a new efficient incremental technique called “magic cyle" that allows to speed up the computation of link loads after each link replacement. This incremental technique was used by all our LS algorithms and we show that it improves both complexity and efficiency in the design and the implementation of our LS algorithms. Our LS algorithm shows good results with different kinds of traffic demand matrices and network topologies.

The promising results obtained in chapter 3 motivated us to deal with the minimization of U_{max} in the DCNs where the MSTP 802.1s is deployed. In Chapter 4, we extended the LS algorithm proposed in Chapter 3 with new heuristics to cope with large DCNs containing many VLANs. Current modern topologies for large data centers containing up to 10K servers were considered for evaluating our LS algorithm. Also, the SNMP data of a private enterprise in the US has been used to simulate the traffic demand matrices in DCNs. The solutions obtained with our LS algorithm could reduce up to 50% U_{max} compared with the solution obtained by 802.1s for the DCNs with 16 VLANs. What we learned from this study is that our LS algorithm for minimizing U_{max} converges very fast since it can approach the best (found) solution within the first 10s of execution time in many tests.

Chapter 5 introduced a bi-objective TE problem with the minimization of the worst-case maximal link utilization (the worst U_{max}) and of the service disruption in case of link failures (the number of link replacements in each spanning tree after the failure of a given link). In this chapter, besides the traditional objective of minimizing U_{max}, our heuristic algorithm provides an efficient link cost configuration technique for each VLAN in the network that can give MSTP an auto-configurable solution with one unique link replacement in each spanning tree after a link failure. Thanks to this link cost configuration technique, the second objective was thus considered as a hard constraint in the minimization of the worst U_{max}. Hence, the solution obtained with our heuristic algorithm guarantees the optimum for the minimization of the service disruption and also shows good performance for minimizing the worst-case maximal link utilization in the network in case of link failures.

The network quality of service is normally expressed by many different objectives to be optimized. Chapter 6 turned to the multi-objective

TE problem in DCNs. We proposed a new design of multi-objective LS algorithms. Our algorithms show good performance for large data centers by integrating the heuristics of each single objective into the search process. There are few existing assessment methods for evaluating the performance of multi-objective TE techniques in DCNs. The state of the art approaches often evaluate their outcomes based on each single objective observation. In this chapter, we introduced a strong assessment method for evaluating the multi-objective overall performance. The non-dominated set of solutions (Pareto front) obtained by our bi-objective algorithms offers the network operators a wide range of good solutions for solving this TE problem in DCNs. Our main contribution in this chapter is that we solved a TE problem in DCNs by following the traditional methodology for multi-objective optimization from problem formulation, algorithm design and development to analysis and evaluation.

Finally, our approximate methods show good performance and strong independency by approaching different aspects of the TE problems in large Ethernet networks with large intances of network topologies and traffic demand matrices.

7.2 Future Work

Nothing is perfect. The main limitation of our approaches in this thesis is that we dealt with off-line TE problems on given static traffic demand matrices. The second limitation is the lack of different switching protocols other than Spanning Tree Protocol and its variants for DCNs. These limitations are due to the academic nature of the research community because of the lack of a real world network for performing the experiments. However, from these non-technical limitations, we would like to state some interesting future research possibilities:

- **Multiple Traffic Demand Matrices:** It would be interesting to extend our local search approaches for solving the same TE problems but with multiple traffic demand matrices instead of one traffic demand matrix for the whole network in case of network with STP and for each VLAN in case of network containing many

VLANs. With the good results obtained in this thesis, we believe that this reasearch direction is promising.

- **Mutiple Link Failures:** In Chapter 5, we considered only each single link failure for a set of given crucial links in the network. But what if more than one link failure occur in the network at a moment? This is also an interesting possibility that could be taken into account for this TE problem.

- **More Objectives:** From the multi-objective viewpoint, many other objectives could be considered for optimizing the performance of the DCNs. We can cite here energy efficiency [CSB+08], network convergence, network delay, etc... In this thesis, we proposed only the approaches for bi-objective TE, but a problem with three or more objectives should also be considered in order to obtain better solutions for improving the performance of the DCNs. Obviously, these problems are more challenging but closer to the need of network operators.

- **Online TE:** The optimization techniques must dynamically adapt the routing when traffic changes in real-time [KKDC05]. This kind of TE problems requires a fast and efficient optimization method, and also credible methods for analysing and preventing the traffic in the networks.

- **OpenFlow:** The emergence of OpenFlow (flow-based) switches [MAB+08] enables DCNs to support more flexible policies rather than standard switching protocols. With OpenFlow switches, we can apply more complex and more intelligent (i.e. multi-path instead of unique shortest cost path with spanning tree) switching schemes on each switch for improving the performance of DCNs.

- **TRILL:** Recently, the TRILL [PED+11] (TRansparent Interconnection of Lots of Links) protocol proposed by R. Perlman allows the ease of configuration of Ethernet while benefitting from the routing techniques provided at Layer 3 with the ability of multicast and broadcast. Although, this protocol still needs time to convince the network operators to replace the Spanning Tree Protocols but it is also a potential direction for the future research.

A

Cycle Load Update Proof

This appendix presents the details of our proof that:
When an edge (s_O, t_O) is replaced by another edge (s_I, t_I) in a spanning tree SP, the load changes only on the cycle C created by adding edge (s_I, t_I) to SP.

Let $C = \{C_1, C_2, .., C_q\}$ be the cycle created by adding e_I into the spanning tree with $e_O = (C_o, C_{o+1})$, $e_I = (C_q, C_1)$ $(1 \leq o < q)$ (see Fig. A.1). We call TI the isolated subtree (excluding root) after removing e_O from SP (see Fig. A.2). We denote s the source node, d the destination node and $TD[s, d]$ the traffic that s wants to send to d. For each pair (s, d) such that $TD[s, d] > 0$, we will prove that the load changes on a link e of the path from s to d if and only if $e \in C$.

PROOF

Case 1: $s, d \in TI$ or $s, d \in SP \setminus TI$
Obviously, the path from s to d is unchanged after removing e_O and adding e_I. Thus the load does not change on the path from s to d.

Case 2: $s \in SP \setminus TI$ and $d \in TI$
2.1. $s \in SP \setminus TI$ and s $\in$ subtree dominated by C_1
We call a vertex u dominates a vertex v in spanning tree SP if u is one of the vertices in the path from v to root.

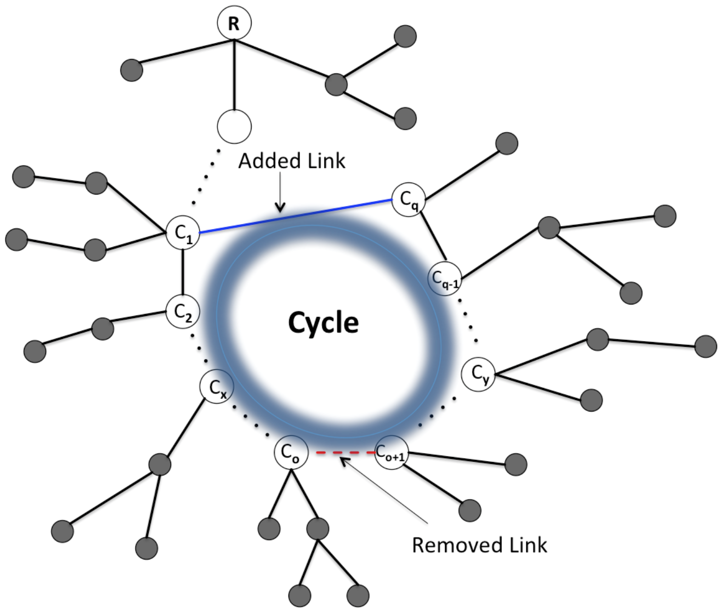


Figure A.1: Cycle created by an edge replacement

Because $s \in SP \setminus TI$ and s $\in$ subtree dominated by C_1, we assume that C_x is the vertex on the cycle C that is closest to s and C_x dominates s (see Fig. A.3). We call C_y the vertex on the cycle C that is closest to d and C_y dominates d.
The old path from s to d before replacing e_O by e_I (the red path in Fig A.3)

$$s \to \ldots \to C_x \to \ldots \to e_O(C_o \to C_{o+1}) \to \ldots \to C_y \to \ldots \to d$$

The new path from s to d after replacing e_O by e_I (the blue path in Fig A.3)

$$s \to \ldots \to C_x \to \ldots \to e_I(C_1 \to C_q) \to \ldots \to C_y \to \ldots \to d$$

To update the load for sending $TD[s, d]$ units of traffic from s to d, we can state that the two sub paths $s \to \ldots \to C_x$ and $C_y \to \ldots \to d$ do not change before and after replacing e_O by e_I. We must remove $TD[s, d]$ units of traffic from the load of each link in $C_x \to \ldots \to$

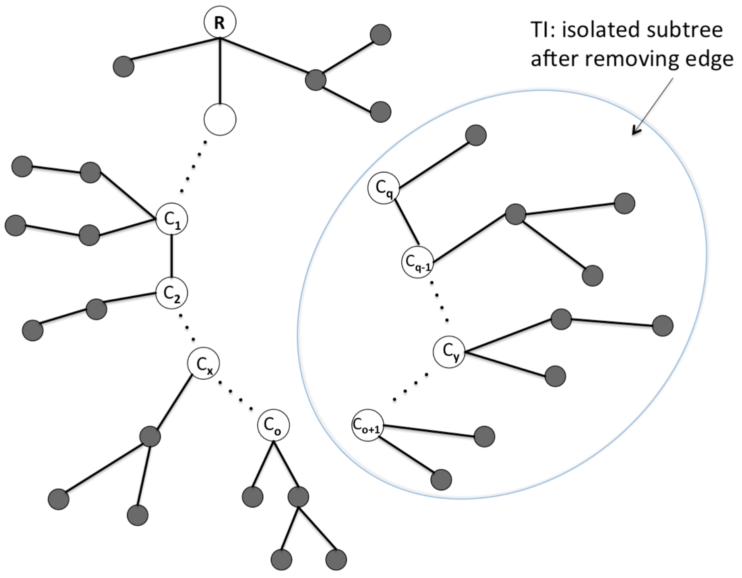


Figure A.2: Isolated tree after removing edge

$e_O(C_o \to C_{o+1}) \to \ldots \to C_y$ (except e_O, its load becomes 0) and add $TD[s,d]$ units of traffic to each link in $C_x \to \ldots \to e_I(C_1 \to C_q) \to \ldots \to C_y$. Obviously, all the links in these two sub paths belong to the cycle C by adding e_I to SP. Hence, in this case, the load changes only on the cycle C.

2.2. $s \in SP \setminus TI$ and s $\notin$ subtree dominated by C_1
In this case, the old path from s to d before replacing e_O by e_I (the red path in Fig A.4)

$$s \to \ldots \to C_1 \to \ldots \to e_O(C_o \to C_{o+1}) \to \ldots \to C_y \to \ldots \to d$$

The new path from s to d after replacing e_O by e_I (the blue path in Fig A.4)

$$s \to \ldots \to e_I(C_1 \to C_q) \to \ldots \to C_y \to \ldots \to d$$

Like in Case 2.1, the two sub paths $s \to \ldots \to C_1$ and $C_y \to \ldots \to d$ do not change before and after replacing e_O by e_I. Thus, we must remove $TD[s,d]$ units of traffic from the load of each link in $C_1 \to \ldots \to$

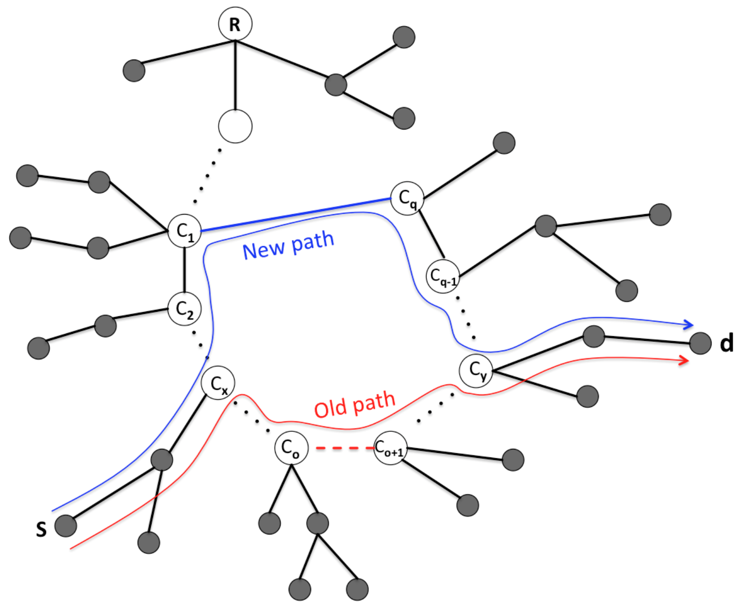


Figure A.3: $s \in SP \setminus TI$ and s $\in$ subtree dominated by C_1

$e_O(C_o \rightarrow C_{o+1}) \rightarrow \ldots \rightarrow C_y$ (except e_O, its load becomes 0) and add $TD[s,d]$ units of traffic to each link in $e_I(C_1 \rightarrow C_q) \rightarrow \ldots \rightarrow C_y$. All these links also belong to the cycle C by adding e_I to SP. So the load changes only on the cycle C.

Case 3: $s \in TI$ and $d \in SP \setminus TI$
Same proof as in Case 2 because in the spanning tree the path from s to d is unique and contains the same links as the path from d to s.

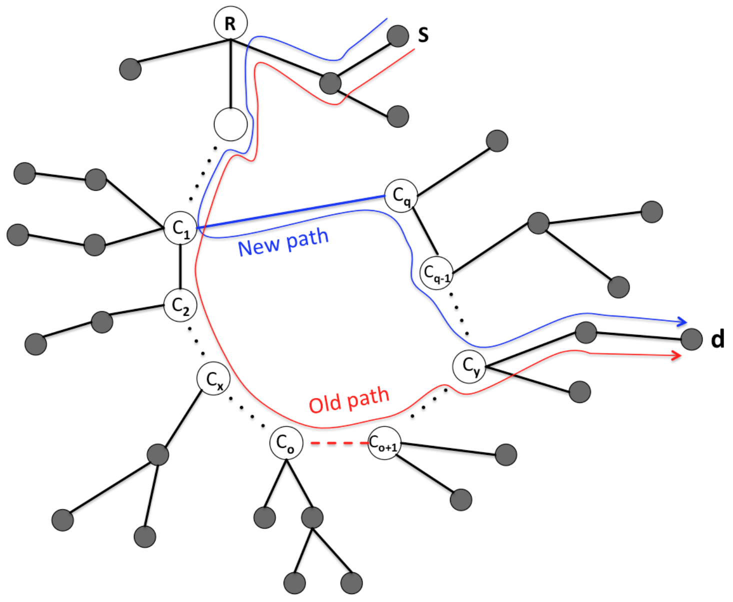


Figure A.4: $s \in SP \setminus TI$ and s $\notin$ subtree dominated by C_1

B

LINK COST UPDATE PROOF

This appendix presents the details of our proof of Property 1 in Chapter 5:
Assume that e_I is chosen as the back-up link of e_O as depicted in Fig. B.1. Let $C = \{C_1, C_2, .., C_q\}$ be the switches on the cycle created by adding e_I into the spanning tree with $e_O = (C_o, C_{o+1})$, $e_I = (C_q, C_1)$ $(1 \leq o < q)$.

If $e_I = (C_q, C_1)$ is chosen as the back-up link for $e_O = (C_o, C_{o+1})$ then each link (C_{o+1}, C_{o+2}), ..., (C_{q-1}, C_q) (the red part of the cycle in Fig. B.1) must set e_I as the back-up link to guarantee the minimum link replacement for its failure.

PROOF

Because e_I is the back-up link for the failure of e_O, then *the unique second shortest path to root from each vertex in $C_{o+1}, C_{o+2}, ..., C_{q-1}, C_q$ must pass via e_I* to ensure that when the failure of e_O occurs, the new spanning tree genrated by MSTP is created by replacing e_O by e_I. (*)
For each link $e = (C_i, C_{i+1})$ with $o + 1 \leq i < q$. We assume that e use a link e' as its back-up link ($e' \neq e_I$). Thus the unique second shortest path to root from C_{i+1} must pass via e' and not via e_I because only one link replacement is allowed. However, C_{i+1} is one of the vertex in $C_{o+1}, C_{o+2}, ..., C_{q-1}, C_q$. This is contradictory to the constraint in (*).

Hence, the Property 1 in Chapter 5 is proved.

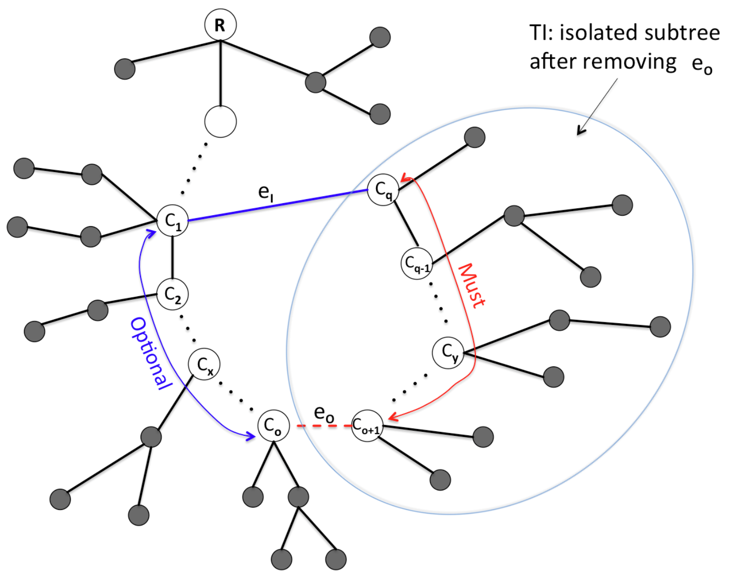


Figure B.1: Link Cost Update

C

Multi-objective TE Result Details

This appendix presents the detail results of our algorithms in Chapter 6.

Table C.1: Performance Comparison U_{max}, $SumL$ - Cloud Geographic

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	23.2	0.0690	0.7459	0.1313	0.8733	0.5612
$Min_{(U_{max},SumL)}$	12	0.9417	0.9722	0.9773	0.9995	0.8704
Min_{SumL}	1	1	0.0852	0.0965	1	0.5109

Table C.2: Performance Comparison U_{max}, $\#Links$ - Cloud Geographic

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	24.6	0.6016	0.4101	0.5117	0.9036	0.4910
$Min_{(U_{max},\#Links)}$	16	1	0.6458	0.9179	1	0.7212
$Min_{\#Links}$	6.4	0.5313	0.2006	0.1614	0.9426	0.4408

Table C.3: Performance Comparison U_{max}, $SumL$ - Cloud Random

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	19.1	0.0628	0.6376	0.1732	0.8721	0.4749
$Min_{(U_{max},SumL)}$	9	0.8778	0.9741	0.9475	0.9999	0.8603
Min_{SumL}	1	1	0.1250	0.1608	1	0.4684

Table C.4: Performance Comparison U_{max}, $\#Links$ - Cloud Random

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	18.6	0.8065	0.3467	0.4602	0.9181	0.4357
$Min_{(U_{max},\#Links)}$	17.3	0.9942	0.6744	0.9276	0.9992	0.6461
$Min_{\#Links}$	13.3	0.4361	0.4762	0.2547	0.8994	0.4142

Table C.5: Performance Comparison U_{max}, $SumL$ - Private Geographic

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	15.7	0.0701	0.6720	0.1048	0.8720	0.4289
$Min_{(U_{max},SumL)}$	16.6	0.9217	0.9858	0.9236	0.9811	0.7713
Min_{SumL}	1.9	1	0.3826	0.3841	1	0.4544

Table C.6: Performance Comparison U_{max}, $\#Links$ - Private Geographic

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	13.3	0.7368	0.3516	0.4340	0.9008	0.4376
$Min_{(U_{max},\#Links)}$	16.3	0.9877	0.7514	0.9253	0.9981	0.6685
$Min_{\#Links}$	10.9	0.1743	0.4662	0.1507	0.8001	0.3642

Table C.7: Performance Comparison U_{max}, $SumL$ - Private Random

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	18.3	0.0601	0.5252	0.1416	0.8846	0.4481
$Min_{(U_{max},SumL)}$	21.1	0.9384	0.9778	0.9694	0.9990	0.9521
Min_{SumL}	1	1	0.0519	0.0782	1	0.4067

Table C.8: Performance Comparison U_{max}, $\#Links$ - Private Random

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	16.3	0.7669	0.3356	0.4071	0.9048	0.4835
$Min_{(U_{max},\#Links)}$	14.2	0.9930	0.7297	0.9407	0.9974	0.7609
$Min_{\#Links}$	9.8	0.2041	0.4296	0.1203	0.8962	0.4159

Table C.9: Performance Comparison U_{max}, $SumL$ - Fat Tree

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	41.7	0.6595	0.8034	0.8424	0.9185	0.5236
$Min_{(U_{max},SumL)}$	15.2	0.8618	0.8011	0.9253	0.9711	0.3834
Min_{SumL}	2.6	0.7308	0.3416	0.2261	0.9604	0.2354

Table C.10: Performance Comparison U_{max}, $\#Links$ - Fat Tree

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	43.7	0.4966	0.1335	0.2133	0.6338	0.7383
$Min_{(U_{max},\#Links)}$	5.3	1	0.5568	0.9148	1	0.8538
$Min_{\#Links}$	4.2	0.9286	0.2790	0.1833	0.9730	0.6955

Table C.11: Performance Comparison U_{max}, $SumL$ - PortLand

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	29.4	0.7211	0.7055	0.9064	0.9674	0.5559
$Min_{(U_{max},SumL)}$	7.6	0.7895	0.5674	0.7476	0.9918	0.2644
Min_{SumL}	1	0.7000	0.0856	0.0347	0.9985	0.2102

Table C.12: Performance Comparison U_{max}, $\#Links$ - PortLand

PF \ Metric	**#Sol**	**Purity**	**HP**	**HP***	**I**	Γ^*
$Min_{U_{max}}$	6.9	0.3043	0.2260	0.1909	0.9226	0.6181
$Min_{(U_{max},\#Links)}$	7.3	1	0.8452	0.9707	1	0.8251
$Min_{\#Links}$	1.2	0	0.1136	0	0.8589	0.5907

Table C.13: Performance Comparison U_{max}, $SumL$ - Expanded Tree 200

PF \ Metric	#Sol	Purity	HP	HP*	I	Γ^*
$Min_{U_{max}}$	22.6	0.3761	0.6619	0.4800	0.7933	0.4978
$Min_{(U_{max},SumL)}$	7.3	0.9315	0.7997	0.9200	0.9494	0.5461
Min_{SumL}	1.9	0.8421	0.3808	0.1747	0.9600	0.4324

Table C.14: Performance Comparison U_{max}, $\#Links$ - Expanded Tree 200

PF \ Metric	#Sol	Purity	HP	HP*	I	Γ^*
$Min_{U_{max}}$	4.7	0.9149	0.0631	0.2435	0.9263	0.8645
$Min_{(U_{max},\#Links)}$	6.1	1	0.6770	0.8814	1	0.8457
$Min_{\#Links}$	3.5	0.1143	0.1976	0.0789	0.7512	0.7742

BIBLIOGRAPHY

[AFLV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM. 42, 47, 92

[AmW99] Daniel O. Awduche and Uunet (mci Worldcom. Mpls and traffic engineering in ip networks. *IEEE Communications Magazine*, 37:42–47, 1999. 31

[AT09] S. Aoufi and D. Tandjaoui. Qos aware multiple spanning tree mechanism in multi-radio multi-channel wireless mesh network. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 296 –302, dec. 2009.

[BAAZ10] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40:92–99, January 2010. 2, 7, 48

[BAM10] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM. 2, 7, 48, 60, 65, 67

[BdJ05] Peter A. N. Bosman and Edwin D. de Jong. Exploiting gradient information in numerical multi–objective evolutionary optimization. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 755–762, New York, NY, USA, 2005. ACM. 26

[CCK64] A. Charnes, R.W. Clower, and K. Kortanek. *Effective Control Through Coherent Decentralization with Preemptive Goals*. Defense Technical Information Center, 1964. 25

[CCMn$^+$01] R. Caballero, E. Cerdá, M. M. Muñoz, L. Rey, and I. M. Stancu-Minasian. Efficient solution concepts and their relations in stochastic multiobjective programming. *J. Optim. Theory Appl.*, 110(1):53–74, 2001. 25

[CJZ06] Wentao Chen, Depeng Jin, and Lieguang Zeng. Design of multiple spanning trees for traffic engineering in metro ethernet. In *Communication Technology, 2006. ICCT '06. International Conference on*, pages 1 –4, nov. 2006. 3, 13, 31, 59

[CMVV11] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and Luís N. Vicente. Direct multisearch for multiobjective optimization. *SIAM Journal on Optimization*, 21(3):1109–1140, 2011. 101, 110, 111

[CSB$^+$08] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 457 –465, april 2008. 17, 102, 126

[DDVH09] Pham Quang Dung, Yves Deville, and Pascal Van Hentenryck. Ls(graph & tree): a local search framework for constraint optimization on graphs and trees. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1402–1407, New York, NY, USA, 2009. ACM. 35, 64 [dSS06] Amaro de Sousa and Gil Soares. Improving load balance of ethernet carrier networks using ieee 802.1s mstp with multiple regions. In *Proceedings of the 5th international IFIP-TC6*, NETWORKING'06, pages 1252–1260, Berlin, Heidelberg, 2006. Springer-Verlag.

[dSS07] A. F. de Sousa and G. Soares. Improving load balance and minimizing service disruption on ethernet networks using ieee 802.1s mstp. In *EuroFGI Workshop on IP QoS and Traffic Contro*. IST Press, 2007. 3, 13, 17, 31, 58, 59, 82, 88, 90, 100

[EZT00] Kalyanmoy Deb Eckart Zitzler and Lothar Thiele. *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*. Evolutionary Computation, 2000. 25

[For00] Bernard Fortz. Internet traffic engineering by optimizing ospf weights. In *in Proc. IEEE INFOCOM*, pages 519–528, 2000. 2, 31, 32, 53

[FS99] J. Fliege and B.F. Svaiter. *Steepest Descent Methods for Multicriteria Optimization*. Informes de matemática. Inst. de Matemática Pura e Aplicada, 1999. 26

[GHJ$^+$09] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM. 65, 66, 92

[GJN11] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 350–361, New York, NY, USA, 2011. ACM. 82

[GL97] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. 22, 39

[GMMO10] Frederic Giroire, Dorian Mazauric, Joanna Moulierac, and Brice Onfroy. Minimizing routing energy consumption: From theoretical to practical results. In *Proceedings of the 2010 IEEE/ACM*, GREENCOM-CPSCOM '10, pages 252–259, Washington, DC, USA, 2010. IEEE Computer Society. 17

[GS00] Ian P. Gent and Barbara M. Smith. Symmetry breaking in constraint programming. In *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000. 35

[HBD$^+$10a] Trong-Viet Ho, O. Bonaventure, Y. Deville, Quang-Dung Pham, and P. Francois. Data set for lsa4stp, 2010. *http://becool.info.ucl.ac.be/page/datas-lsa4stp*. 50

[HBD$^+$10b] Trong-Viet Ho, O. Bonaventure, Y. Deville, Quang-Dung Pham, and P. Francois. Using local search for traffic engineering in switched ethernet networks. In *Teletraffic Congress (ITC), 2010 22nd International*, pages 1 –8, sept. 2010. 29

[HDB12] Trong-Viet Ho, Y. Deville, and O Bonaventure. Data set for multi-objective traffic engineering, 2012. *http://becool.info.ucl.ac.be/resources/multi-objective-traffic-engineering-data-center-networks*. 115

[HDBF11a] Trong-Viet Ho, Y. Deville, O. Bonaventure, and P. Francois. Data set for lsa4mstp, 2011. *http://becool.info.ucl.ac.be/page/ traffic-engineering-multiple-spanning-tree-protocol-large-data-centers.* 71

[HDBF11b] Trong-Viet Ho, Y. Deville, O. Bonaventure, and P. Francois. Traffic engineering for multiple spanning tree protocol in large data centers. In *Teletraffic Congress (ITC), 2011 23rd International*, pages 23 –30, sept. 2011. 57

[HFDB09] Trong-Viet Ho, Pierre Francois, Yves Deville, and Olivier Bonaventure. Implementation of a traffic engineering technique that preserves IP Fast Reroute in COMET. In Augustin Chaintreau and Clemence Magnien, editors, *AlgoTel*, Carry-Le-Rouet, France, 2009. 29, 31, 50, 53

[HSK06] Ken Harada, Jun Sakuma, and Shigenobu Kobayashi. Local search for multiobjective function optimization: pareto descent method. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 659–666, New York, NY, USA, 2006. ACM. 27

[Hu74] T. C. Hu. Optimum communication spanning trees. *SIAM J. Comput.*, 3(3):188–195, 1974. 16, 103

[HZC06] Xiaoming He, Mingying Zhu, and Qingxin Chu. Traffic engineering for metro ethernet based on multiple spanning trees. In *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, page 97, 2006. 3, 13, 26, 31, 58, 100

[KC00] Joshua Knowles and David Corne. M-paes: A memetic algorithm for multiobjective optimization. pages 325–332. IEEE Press, 2000. 26

[KdW06] I. Kim and O. de Weck. Adaptive weighted sum method for multiobjective optimization: a new method for Pareto front generation. *Structural and Multidisciplinary Optimization*, 31:105–116, February 2006. 25

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. 22

[KKDC05] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: responsive yet stable traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 35(4):253–264, August 2005. 12, 126

[KSG$^+$09] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM. 7, 48

[LBJT07] Arnaud Liefooghe, Matthieu Basseur, Laetitia Jourdan, and El-Ghazali Talbi. Combinatorial optimization of stochastic multi-objective problems: An application to the flow-shop scheduling problem. 4403:457–471, 2007. 25

[LCP11] Jung Song Lee, Lim Cheon Choi, and Soon Cheol Park. Multi-objective genetic algorithms, nsga-ii and spea2, for document clustering. In *Software Engineering, Business Continuity, and Education*, volume 257 of *Communications in Computer and Information Science*, pages 219–227. Springer Berlin Heidelberg, 2011. 25

[lin99] Multi-objective linear programming. In Freerk A. Lootsma, Panos M. Pardalos, and Donald W. Hearn, editors, *Multi-Criteria Decision Analysis via Ratio and Difference Judgement*, volume 29 of *Applied Optimization*, pages 229–257. Springer US, 1999. 25

[LYD$^+$03] Yujin Lim, Heeyeol Yu, Shirshanka Das, Scott Seongwook Lee, and Mario Gerla. Qos-aware multiple spanning tree mechanism over a bridged lan environment. In *Proceedings IEEE GLOBECOM 2003*, 2003. 3, 13, 26, 31, 58, 100

[MAB$^+$08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. 12, 126

[MAK07] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical Aspects of Local Search*. Springer, 2007. 20, 26

[Med06] A. Meddeb. Ngl01-3: Multiple spanning tree generation and mapping algorithms for carrier class ethernets. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1 –5, 27 2006-dec. 1 2006. 3, 13, 31, 58

[MSBR09a] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. Energy aware network operations. In *INFOCOM Workshops 2009, IEEE*, pages 1 –6, april 2009. 17

[MSBR09b] Priya Mahadevan, Puneet Sharma, Sujata Banerjee, and Parthasarathy Ranganathan. A power benchmarking framework for network devices. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag. 17

[MSS09] G. Mirjalily, F.A. Sigari, and R. Saadat. Best multiple spanning tree in metro ethernet networks. In *Computer and Electrical Engineering, 2009. ICCEE '09. Second International Conference on*, volume 2, pages 117 –121, dec. 2009. 3, 13, 31, 59

[NMPF$^+$09] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, August 2009. 42, 46, 92

[NP09] Antonio Nucci and Konstantina Papagiannaki. *Design, Measurement and Management of Large-Scale IP Networks: Bridging the Gap Between Theory and Practice*. Cambridge University Press, 2009. 31, 53

[NRR11] E.R. Naganathan, S. Rajagopalan, and P.H. Raj. Traffic flow analysis model based routing protocol for multiprotocol label switching network. *Computer Science, 7: 1674-1678*, 2011. 1

[NY06] Hirotaka Nakayama and Yeboon Yun. Generating support vector machines using multi-objective optimization and goal programming. In Yaochu Jin, editor, *Multi-Objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*, pages 173–198. Springer Berlin / Heidelberg, 2006. 25

[PED$^+$11] R. Perlman, D. Eastlake, D. Dutt, S. Gai, and A. Ghanwani. Routing bridges (rbridges): Base protocol specification. July 2011. 126

[Per99] Radia Perlman. *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols (2nd Edition).* Addison-Wesley, 1999. 9

[Per04] Radia J. Perlman. Rbridges: Transparent Routing. In *INFOCOM*, 2004. 53

[Per10] Radia J. et al. Perlman. Rbridges: Base protocol specificationâĂİ. Jan. 2010. 53

[PNM$^+$05] M. Padmaraj, S. Nair, M. Marchetti, G. Chiruvolu, M. Ali, and A. Ge. Metro Ethernet traffic engineering based on optimal multiple spanning trees. March 2005. 3, 13, 31, 59

[RS95] Rajiv Ramaswami and Kumar N. Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Trans. Netw.*, 3(5):489–500, October 1995. 31

[SD94] N. Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, September 1994. 25

[SdSA$^+$09] D. Santos, A. de Sousa, F. Alvelos, M. Dzida, M. Pioro, and M. Zagozdzon. Traffic engineering of multiple spanning tree routing networks: the load balancing case. In *Next Generation Internet Networks, 2009. NGI '09*, pages 1 –8, july 2009. 3, 13, 31, 58, 59

[SdSA$^+$11] Dorabella Santos, Amaro de Sousa, Filipe Alvelos, Mateusz Dzida, and Michal Pioro. Optimization of link load balancing in multiple spanning tree routing networks. *Telecommunication Systems*, 48:109–124, 2011. 10.1007/s11235-010-9337-8. 3, 13, 26, 31, 58, 59

[SGD05] Ashwin Sridharan, Roch Guérin, and Christophe Diot. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks. *IEEE/ACM Trans. Netw.*, 13(2):234–247, April 2005. 31, 53

[Soc98] IEEE Computer Society. Information technology - telecommunications and information exchange between systems local and metropolitan area networks common specifications part 3: Media access control bridges. *IEEE Standard 802.1D*, 1998. 9, 30, 57, 93, 104

[Soc01] IEEE Computer Society. Rapid spanning tree configuration. *IEEE Standard 802.1W*, 2001. 10

[Soc02] IEEE Computer Society. Virtual bridged local area networks - amendment 3: Multiple spanning trees. *IEEE Standard 802.1S*, 2002. 10, 31, 57, 58, 101

[Soc06] IEEE Computer Society. Virtual local area networks. *IEEE Standard 802.1Q*, 2006. 10, 58

[SS83] H. D. Sherali and A. L. Soyster. Preemptive and nonpreemptive multi-objective programming: Relationship and counterexamples. *Journal of Optimization Theory and Applications*, 39:173–186, 1983. 25

[Sys10] Cisco Systems. Cisco data center infrastructure 2.5 design guide, December 2010. *http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf*. 8, 65, 66, 92

[VM05] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT Press, London, England, 2005. 35

[YST85] Hirotaka Nakayama Yoshikazu Sawaragi and Tetsuzo Tanino. *Theory of multiobjective optimization*. Academic Press, 1985. 23, 100

[Zel74] M. Zeleny. *Linear multiobjective programming*. Lecture notes in economics and mathematical systems. Springer-Verlag, 1974. 25

[ZRDG03] Yin Zhang, Matthew Roughan, Nick Duffield, and Albert Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pages 206–217, New York, NY, USA, 2003. ACM. 68

[ZT98] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. pages 292–301. Springer, 1998. 101, 110, 113

[ZTL$^+$03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, May 2003. 101, 110, 115