

ALGORITHMS FOR THE SURVIVABLE TELECOMMUNICATIONS NETWORK DESIGN PROBLEM UNDER DEDICATED PROTECTION

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Pelin Damcı

July, 2010

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Oya Ekin Karařan (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Hande Yaman

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. İbrahim K rpeođlu

Approved for the Institute of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Institute

ABSTRACT

ALGORITHMS FOR THE SURVIVABLE TELECOMMUNICATIONS NETWORK DESIGN PROBLEM UNDER DEDICATED PROTECTION

Pelin Damcı

M.S. in Industrial Engineering

Supervisor: Assoc. Prof. Dr. Oya Ekin Karaşan

July, 2010

This thesis presents algorithms to solve a survivable network design problem arising in telecommunications networks. As a design problem, we seek to find 2-edge disjoint paths between every potential origin destination pair such that the fixed costs of installing edges and the routing costs are jointly minimized. Despite the fact that the survivable network design literature is vast, the particular problem at hand incorporating fixed and variable edge costs as well as different cost structures on the two paths has not been studied. Initially, an IP model addressing the proposed problem is developed. In order to solve problems of higher dimensions, different heuristic algorithms are designed and results of a computational study on a large bed of problem instances are reported.

Keywords: Survivable network design, Primary and secondary paths, Dedicated protection.

ÖZET

ADANMIŞ KORUMALI GÜVENİLİR HABERLEŞME AĞLARI İÇİN ALGORİTMALAR

Pelin Damcı

Endüstri Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Doç. Dr. Oya Ekin Karışan

Temmuz, 2010

Bu tez, güvenilir haberleşme ağları tasarımı problemlerini çözmek için algoritmalar sunmaktadır. Amacımız her bir kaynak ve hedef ikilisi için 2-ayrıt yol bulan ve aynı zamanda ayrıt kullanmak için verilen sabit giderleri ve yol atama maliyetlerini enküçülten bir tasarım elde etmektir. Her ne kadar güvenilir haberleşme ağları ile ilgili geniş bir teknik yazın kaynakçası olsa da, bahsettiğimiz her bir ayrıt için sabit giderleri, rotalama maliyetlerini ve her bir yol için farklı maliyet yapısını göz önünde bulunduran problem daha önce çalışılmamıştır. İlk olarak, bu problem için bir tamsayılı programlama modeli geliştirilmiştir. Büyük ölçekli problemleri çözebilmek için farklı sezgisel algoritmalar tasarlanmıştır ve bu algoritmaların hesaplama sonuçları çok sayıda örnek için rapor edilmiştir.

Anahtar sözcükler: Güvenilir ağların tasarımı, Birincil ve ikincil yollar, Adanmış koruma.

To my parents and grandfather Ali Rıza Özbek...

Acknowledgement

First and foremost, I feel very lucky to have had worked with both of my Professors; Assoc. Prof. Dr. Oya Ekin Karařan and Assoc. Prof. Hande Yaman. Therefore, here I would like to express my gratitude to both of them. Throughout the two years I spent at Bilkent University as a M.S. student I have learned from my Professors how to balance my work both in and out of class.

Furthermore, my thanks and appreciation goes to my thesis committee member, Asst. Prof. Dr. İbrahim K rpeođlu, for his valuable time and reviews of this thesis.

I am grateful to T BİTAK for the financial support they provided during my research.

My mother and father, G l in Damcı and Neziĥ Damcı have supported me throughout my education life. They made me realize I could do whatever I set my mind to in life. I'd like to thank them for all they are, and all they have done for me.

I thank my fiancée, Mehmet Can Kurt for his amazing support during my six years at Bilkent University.

Finally, I would like to send my best wishes and thanks to my friends who have given me support throughout my two years of graduate study. Gonca Aydođdu, İlker Tufan, Ezgi Demirci, G k e Akın, Korhan Aras, Hatice  alık, Ece Zeliha Demirci, G lřah Han erliođulları, Ceyda Kırık ı and Bařak Renkliođlu. I feel very lucky that you were by my side! I am also thankful to all other friends that I failed to mention here.

Contents

1	Introduction	1
2	Literature Review	6
3	Mathematical Model	12
4	Heuristic Algorithms	17
4.1	Suurballe's Algorithm	18
4.2	Construction Heuristics	19
4.2.1	Two-Step Algorithm	19
4.2.2	One-Step Algorithm	23
4.3	Improvement Heuristics	26
4.3.1	IP Based Heuristic	26
4.3.2	Edge Deletion	27
4.3.3	Edge Addition	27
4.3.4	Cycle Algorithm	28

5	Numerical Results of Algorithms	32
5.1	Test Instances	32
5.1.1	Node Number (V)	33
5.1.2	Edge Number (E)	33
5.1.3	Primary and Secondary Path Costs (ce_{ij}^1 and ce_{ij}^2)	33
5.1.4	Fixed Cost (f_{ij})	33
5.1.5	Demand (d)	34
5.2	Results of Test Instances	34
5.2.1	One-step and One-step-2 Algorithm Comparison	35
5.2.2	Edge Deletion Heuristic and Cycle Algorithm Comparison	36
5.2.3	Summary of Our Findings	39
6	Conclusion	47
A	Results of Computational Studies	51

List of Figures

1.1	2-edge disjoint paths from s to t : Path 1: $s-1-3-5-t$, Path 2: $s-2-4-t$	5
4.1	An example which shows how Suurballe's algorithm works	20
4.2	Steps of Edge Deletion Heuristic	27
4.3	Steps of Edge Addition Heuristic	28
4.4	An example for a single commodity i and j which shows how cycle algorithm works	29
5.1	One-step and One-step-2 Algorithm Results for node $\# = 30$. . .	37
5.2	One-step and One-step-2 Algorithm Results for node $\# = 40$. . .	38
5.3	Edge Deletion Heuristic and Cycle Algorithm Results for node $\# = 30$ and Comparison Results for One-step and One-step-2 Algorithms as the Construction Heuristics	40
5.4	Edge Deletion Heuristic and Cycle Algorithm Results for node $\# = 40$ and Comparison Results for One-step and One-step-2 Algorithms as the Construction Heuristics	41
5.5	Edge Deletion Heuristic and Cycle Algorithm Results for node $\# = 30$	42

5.6	Edge Deletion Heuristic and Cycle Algorithm Results for node # = 40	43
5.7	One-step and One-step-2 Algorithm: Results for All Test Instances	45
5.8	Edge Deletion Heuristic and Cycle Algorithm: Results for All Test Instances	46

List of Tables

3.1	Notation	13
A.1	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$	53
A.2	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$	54
A.3	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$	55
A.4	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$	56
A.5	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$	57
A.6	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$	58

A.7	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 1, Demand $\in \{1, 10\}$	59
A.8	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 10, Demand $\in \{1, 10\}$	60
A.9	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 100, Demand $\in \{1, 10\}$	61
A.10	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 1, Demand $\in \{1, 10\}$	62
A.11	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 10, Demand $\in \{1, 10\}$	63
A.12	Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 100, Demand $\in \{1, 10\}$	64
A.13	Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$	65
A.14	Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$	66
A.15	Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$	67

A.16 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$	68
A.17 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$	69
A.18 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$	70
A.19 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 10\}$	71
A.20 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 10\}$	72
A.21 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 10\}$	73
A.22 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 10\}$	74
A.23 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 10\}$	75
A.24 Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 10\}$	76

A.25 Computational Results of the One-Step Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$	77
A.26 Computational Results of the One-Step Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$	78
A.27 Computational Results of the One-Step Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$	79
A.28 Computational Results of the One-Step Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$	80
A.29 Computational Results of the One-Step Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$	81
A.30 Computational Results of the One-Step Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$	82
A.31 Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$	83
A.32 Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$	84
A.33 Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$	85
A.34 Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$	86
A.35 Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$	87
A.36 Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$	88

A.37 Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 10\}$	89
A.38 Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 10\}$	90
A.39 Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 10\}$	91
A.40 Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 10\}$	92
A.41 Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 10\}$	93
A.42 Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 10\}$	94

Chapter 1

Introduction

Networks are being used in many settings to model and solve problems. A number of problems that can be found in everyday life using networks are: Finding shortest paths, designing telecommunications networks, speeding up Internet, balancing the traffic on highways etc. Since the 90's, an increasing number of telecommunications networks are being used. Especially, with the vast development of the Internet, and the need to transmit more data, design of survivable and properly capacitated networks have become imperative. Survivability is a keyword for today's telecommunications networks due to the domination of telecommunications systems that have invaded consumers' lives in every way. Hence, survivability is a critical design constraint for high-speed networks that satisfy the users. There are different types of protection schemes. However, the general idea is to connect source and destination pairs with more than a single path. By this way, if a failure on a path occurs another path can become active and the data transfer to the destination can be made safely. Equipment failures may occur due to construction or due to destructive natural events, such as earthquakes, tsunamis, tornadoes etc. Since, repairing an equipment in a short time may not be possible, the use of another path during the mending of the failure will be necessary for the transfer of data between the source and the destination to continue.

How to choose the paths between source and destination pairs depends on the

severity of the survivability requirement. In general, node or link failures may occur so the paths can be either node-disjoint or edge-disjoint. Any number of disjoint paths can be considered from a source to a destination. But, finding and using disjoint paths is costly therefore a balance between survivability and total costs needs to be considered. In the literature two types of paths are referred to; primary(working) and secondary(backup) paths and most research focuses on the recovery from a single link or node failure. In other words, one failure is repaired before another failure occurs. Nonetheless, multiple failures in a realistic network may occur but this subject is beyond the scope of this thesis.

There are two types of protection schemes for using primary and secondary paths as survivability measures; dedicated protection and shared protection. In dedicated protection, a spare capacity is available such that if a destination point suffers a failure due to the spare capacity it is guaranteed that there will be available resources to recover from the failure, assuming that the secondary resources have not failed. There are two types of categories of dedicated protection. In 1+1 dedicated protection, both the primary and the secondary path is active and the circuitry in the network chooses the better connection. On the contrary, in 1:1 dedicated protection only the primary path is active until a failure occurs in the network only then the secondary path is used. After the failure is overcome the primary path can be used again or the connection may continue to use the secondary path. The advantage of 1+1 dedicated protection is that the recovery from a failure can be nearly immediate. But, 1:1 dedicated protection is slower since the transmit must start over from the secondary path. Since 1+1 dedicated protection has both the primary and secondary paths active this type of approach requires more equipment which may be costly. For both 1+1 and 1:1 dedicated protection a spare capacity is needed which is considered as a downside of the dedicated protection scheme due to its cost. Shared dedication protection addresses this downside by making the spare capacity available for more than one primary path. There is a restriction to which primary paths can share the spare capacity; they cannot have links or nodes in common.

While finding node-disjoint or edge-disjoint paths the total costs incurred by using the edges are considered. The edges have costs such that if 1 unit of demand

is sent through that edge the cost of that specific edge is added to the total cost. This is called the minsum problem which consists of finding k disjoint paths between two distinct nodes, a source and a destination such that the sum of the cost of the routes is minimum [5]. A polynomial running time algorithm developed for the edge-disjoint problem by Suurballe and Tarjan [12] solves the problem of finding 2-edge disjoint paths to optimality when the objective is the minimization of the sum of the costs of the used edges on both paths. However, Suurballe's algorithm finds an optimal solution for only a single source and destination pair and when there is a single cost for each edge. According to the requirements of survivability and the objective function the problem can become NP-hard which causes the researchers to focus on different heuristic algorithms. In many real life problems which fall into the category of survivable network design there is a relationship between the two costs $c_p(e)$, $c_s(e)$ for each edge e , where the former cost is used to compute the cost of a primary path while the latter is used for secondary path computation. This relationship is typically characterized in terms of a coefficient α such that $0 < \alpha < 1$ and $c_s(e) = \alpha c_p(e)$. However, the costs $c_p(e)$, $c_s(e)$ can also be arbitrary. For the special case of $c_s(e) = \alpha c_p(e)$ for all edges e , the problem of minimizing the total costs incurred is known to be NP-hard for directed graphs, i.e. graphs in which links have directions. This result holds whether the paths are required to be node or edge disjoint[3]. The node-disjoint and edge-disjoint paths problem for undirected graphs is also known to be NP-hard according to Xu et al. [14]. However, Bhatia et al. show that Xu et al.'s proof for the edge-disjoint problem in undirected graphs is flawed.

Other than the minsum problem, there is also a min-max version of the problem. This problem minimizes the cost of the most expensive of the selected routes. Min-max version is much more difficult, Li et al. [9] showed that the min-max problem is strongly NP-complete even when $k = 2$ for the four possible variants of the problem; edge-disjoint, node-disjoint and the network is either directed or undirected.

In this thesis our aim is to solve a network design problem with requirements that define the survivability level along with different cost structures while minimizing the total cost. We seek 2-edge disjoint paths for every possible origin

destination pair. We are given a graph $G = (V, E)$, where V represents the node set and E represents the edge set. There is a fixed cost, i.e., a cost for opening (or activating) an edge and two variable costs for an edge. Fixed cost is incurred once if edge is used. Two variable costs are costs for sending 1 unit of flow from that edge. There are two variable costs since one of the costs ce_{ij}^1 is incurred if that edge $\{i, j\} \in E$ is used along a primary path for a source-destination pair and the other cost ce_{ij}^2 is incurred if edge $\{i, j\} \in E$ is used along a secondary path for a source-destination pair. These two costs are also referred to as dual edge costs since there are two costs for each edge. We assume that the relationship between primary and secondary path costs is $c_s(e) = 1/2c_p(e)$. In the literature, the cost of using secondary paths is generally accepted as lower than their primary counterparts. This is because in normal circumstances the primary paths are used and only if some damage occurs in a primary path the secondary path is utilized for a source-destination pair. Any source-destination pair has a demand that needs to be satisfied. Throughout this thesis all possible source and destination pairs are assumed to exist. Figure 1.1 represents 2-edge disjoint paths for a source-destination pair, $s - t$. The first path is $s \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow t$ and the second path is $s \rightarrow 2 \rightarrow 4 \rightarrow t$. One may notice that there are only 2-edge disjoint paths from s to t hence, for any $k > 2$ this example will be infeasible. However, more than 1 different pair of edge-disjoint paths can be found. For example, the first path can be $s \rightarrow 1 \rightarrow 3 \rightarrow t$ and the second path can be $s \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow t$.

The rest of the thesis is organized as follows. Chapter 2 contains the literature review for survivable network design. In Chapter 3 an IP model which solves the 2-edge disjoint network problem with fixed and dual edge costs is presented. In Chapter 4, the specific details of the algorithms used to solve the problem are explained. In Chapter 5, the numerical results obtained from both the IP model and the algorithms are provided along with interpretations of the results. Finally, the thesis is summarized in Chapter 6 and possible future work is also discussed. For detailed numerical results the reader can review the Appendix.

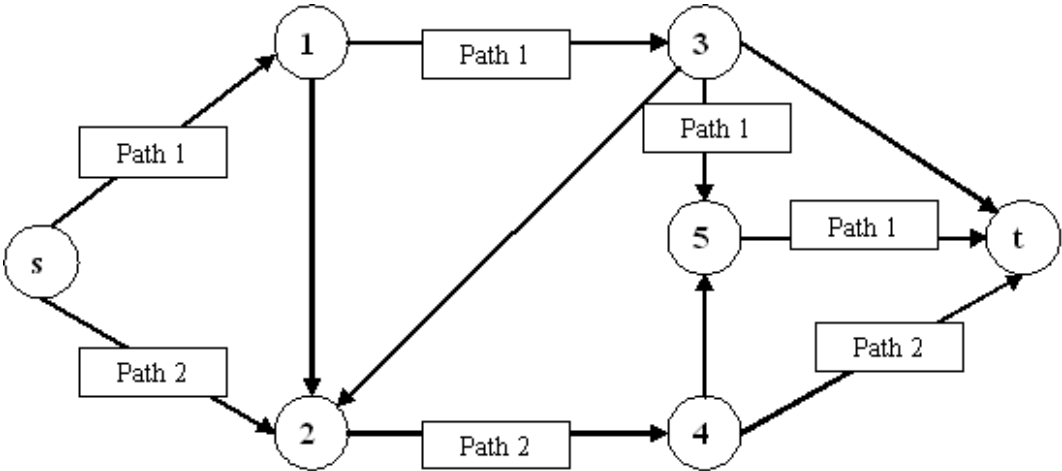


Figure 1.1: 2-edge disjoint paths from s to t : Path 1: $s-1-3-5-t$, Path 2: $s-2-4-t$

Chapter 2

Literature Review

Survivability has become a major issue in telecommunications networks with the emerging need to transfer more data compared to earlier decades. Consumers desire a satisfying service that is without failure at all times. This requirement can be met via utilizing more than one route between a source and a destination pair to transfer data. Having an additional route, referred to as the back up or secondary path will provide a protection against a failure caused by a destruction in the primary path. By assumption a single node or link failure can happen. Therefore, if a problem occurs in transferring data that uses the primary path automatically the data transfer is continued by utilizing the secondary path until the complication in the primary path is fixed. To use the secondary path without any difficulties the primary and secondary path must be node or edge disjoint (having no common nodes or no common edges) that is, if one of the paths is out of order the other one will be ready to continue the data transfer.

The problem of finding “disjoint paths” is being studied since late 1950’s. Developments in MIP models, heuristic and exact algorithms can be seen throughout five decades. Research related to finding “disjoint paths” is divided according to different constraints since several types of equipment failure may occur in a network and interrupt traffic along paths. Hence, one of the major responsibilities of “disjoint paths” between a given source and sink is to increase reliability in communication networks. However, increasing reliability “too much” may be

costly according to given network parameters. Therefore, a compromise between reliability and total cost needs to be achieved for a desirable output. Given a brief description of the general problem, the following paragraphs analyze the literature on “disjoint paths” in more depth.

In Suurballe’s paper [13] the problem of finding K node-disjoint paths with minimum total cost is presented. The total cost includes the summation of individual arc lengths used on paths between a source and a destination. Suurballe [13] describes a labeling algorithm involving K shortest path iterations. The idea presented by the author gives a polynomial time algorithm. Furthermore, with slight modifications as discussed in [12] the algorithm can also be used for finding edge-disjoint paths. Bhandari’s algorithm [2] which is a slight variation of Suurballe’s algorithm also achieves the same results as Suurballe’s algorithm. Both Suurballe’s algorithm and Bhandari’s algorithm provide optimal results to the problem of finding a pair of edge-disjoint paths for a single source and destination pair. The problem solved by Suurballe and Bhandari differ from our problem in several ways. First of all, we assume that all possible source and destination pairs exist in a given graph G . In addition, we consider two different path costs ce_{ij}^1 and ce_{ij}^2 for each edge $\{i, j\}$ and fixed cost f_{ij} of activating an edge $\{i, j\}$.

Li et al. [10] consider a different problem compared to Suurballe since for a network $G = (V, E)$ with source and sink nodes there are k different costs on every edge. Li et al. describe their problem as a minsum problem, where j^{th} edge-cost is associated with the j^{th} path. They analyze several variants of the problem; node-disjoint or edge-disjoint problems with directed or undirected networks. Li et al. claim that all four versions of the problem are NP-complete even when $k = 2$, for arbitrary primary and secondary path costs, however, Bhatia et al. [3] show that when, $ce_{ij}^1 < ce_{ij}^2$, Li et al.’s. NP hardness results do not extend to this case. Li et al. present polynomial time heuristics and an algorithm for their proposed problem. The first heuristic they describe is a heuristic in which a function f of the k costs on each arc that “averages” the individual costs is defined. However, this “averages” concept is a function determined by the users of the algorithm so it may or may not be the same as the customary meaning of taking the average of k numbers. After computing this function f of k different

costs for each arc a Minimum Cost Network Flow (MCNF) problem is solved such that the supply at the source node is k , the demand at the sink node is k , all other nodes have supply/demand equal to 0 and edges have capacity of 1. The second heuristic discussed in the paper arranges the edge costs according to the customary meaning of average, so it takes the average of the k different costs without creating a function f . Although Li et al. consider different path costs for edges their problem is fundamentally different from our problem since they do not consider fixed costs for activating edges.

Bhatia et al. [3] point out that the cost metric used for primary and secondary paths differ in some settings and in others they are somehow related to each other. More precisely, one of the costs may be a multiple of the other. The problem considered by Bhatia et al. is to find a pair of edge or node-disjoint paths of minimum cost where the costs of primary path is the total cost of the edges used on the paths while the cost for the secondary path is α times the sum of the cost of the links used on the path, where $\alpha < 1$. This study is of great importance to this thesis where $\alpha = 1/2$ for all of the test instances that are present in Chapter 5. Bhatia et al. argue that a simple algorithm achieves an approximation ratio of $O(1/\alpha)$ for the proposed problem. They also consider the four versions of the problem that are previously described in the above paragraph. The approximation algorithm they mention is Suurballe's algorithm which runs in polynomial time. They prove that this algorithm is a $1/2 + 1/\alpha$ approximation algorithm for their problem. They conclude by saying that if α is a fixed constant, as in this thesis, the hardness of the problem is still an open question. The problem that Bhatia et al. consider is the closest one to our problem in survivable network design literature. However, like other papers they do not consider fixed costs for activating edges.

A recent study conducted by Gomes et al. [5] like Li et al. [10] also analyzes the problem of calculating k disjoint paths from a source to a destination (two distinct nodes) in which there are k arbitrary costs on every edge and the total cost is minimized. Even when $k = 2$ this problem is NP-complete since the costs on edges are arbitrary. The authors refer to the networks as dual arc cost networks when $k = 2$. They propose an exact algorithm that finds 2 disjoint

paths for source and destination pairs when the network has dual arc costs. The exactness of the algorithm they describe results from the fact that it allows the calculation of optimal solutions by using a condition to satisfy the optimality. The algorithm is based on calculating upper and lower bounds on the optimal cost. Two alternative problems can also be solved by slightly modifying their proposed algorithm. These problems are finding node-disjoint paths and disjoint paths with length constraints. The authors claim that their exact algorithm can solve any instance to optimality if memory and CPU times were unlimited. They present test instances with up to 1000 nodes and when the number of arcs are 3 or 4 times the number of nodes in the network. The worst case complexity of the algorithm is $O(n^3(u + v) + n(u + v)^2)$, where n is the number of nodes and $u + v$ is the number of generated shortest paths. The study presented by Gomes et al. in [5] is an extension to ideas presented in [4]. Although the algorithms provided in the two papers differ, the basic approach used to find the optimal solutions remains the same. The exact algorithm presented by Gomes et al. [5] is not utilized in this thesis due to its high complexity and memory usage. In addition, the algorithm described by the authors do not take into consideration fixed costs for activating edges.

Ho et al. [7] propose an Integer Linear Program (ILP) and two heuristics called Iterative Two-Step-Approach (ITSA) and Maximum Likelihood Relaxation (MLR) to solve the least-cost primary and secondary path-pair (in terms of the sum of the total cost). The authors use the shared protection scheme while solving the problem. Recall that in shared protection scheme a spare capacity is available such that if a destination point suffers a failure due to the spare capacity it is guaranteed that there will be available resources to recover from the failure, assuming that the secondary resources have not failed. In contrast to dedicated protection scheme this spare capacity is available for more than one primary path assuming that the primary paths in consideration do not share a link or a node. The ITSA heuristic enumerates and inspects all of the k -shortest paths as the primary path. Although ITSA provides better results in terms of the proximity to the optimal solution, the computational complexity becomes a bottleneck for larger problems in terms of node or link numbers. The other

heuristic, MLR is a modified version of the Dijkstra's algorithm [1] and yields polynomial time complexity. To explain it in more detail, MLR considers finding the secondary path during the calculations for finding the primary path. However, since MLR yields a polynomial time complexity the results obtained from it are not as satisfying as ITSA's computational results. Networks with up to 100 nodes have been tested in the paper.

Another version of finding disjoint paths problem is to maximize the number of disjoint paths between a source and a destination. An extension to this problem is that length of every path is bounded by a given value, p . Itai et al. [8] analyze the complexity of this problem, while Perl and Ronen [11] present a polynomial time heuristic algorithm for any given bound value, p . For the test instances used in the paper, they prove that when $p \leq 5$ optimal solutions are found and when $p \geq 5$ solution values are in proximity to the optimal solution.

In addition to minimizing the cost of finding disjoint paths for single source and destination problems, several different source and destinations can also be added to the problem. However, having several source and destinations increases the complexity of the problem. Depending on the context of the problem there may be several destination points and one single source or vice versa. But any number of source and destination pairs is also possible.

One other problem differing from the previously mentioned settings is presented by Guruswami et al. in [6]. The specific problem at hand is finding a maximum number of length bounded edge-disjoint paths between any given source and destination pairs. The authors show an analysis of the approximability of the proposed problem. Having presented their analysis, an $O(\sqrt{m})$ time approximation algorithm to solve the maximum edge-disjoint path problem is also provided.

The contribution of this thesis to the survivable network design literature is as follows: A new 2-edge connected network design problem is introduced. The task is to find 2-edge disjoint paths for every possible source to destination in the presence of fixed costs for edges and different routing costs for primary and secondary paths. First of all, the costs of primary and secondary paths

are different but related to each other since, $ce_{ij}^1 = 2ce_{ij}^2$ (primary path costs is 2 times the secondary path costs). Although there are some studies on different path costs for edges, as Bhatia et al. [3] point out when $\alpha ce_{ij}^1 = ce_{ij}^2$ and α is a constant the question of whether the problem is NP-hard or not is still an open question. However, in addition to routing costs, ce_{ij}^1 and ce_{ij}^2 , we also consider fixed costs f_{ij} for each edge $\{i, j\} \in E$.

In the next chapter, the IP model for the proposed problem is presented.

Chapter 3

Mathematical Model

In this section the IP model for our problem, namely, the Survivable Network Design-Dedicated Protection (SND-DP) is introduced along with some analysis. Given a network, the task is to find 2-edge disjoint paths for every commodity in the network such that primary and secondary paths have different edge costs and a fixed cost for opening an edge for usage is encountered. Although a network is assumed to be available at hand, using an edge for the first time or activating it requires a fixed cost. The outcome of the model will produce two paths that have no common edges for each commodity. The objective is to minimize the total costs.

Given a graph $G = (V, E)$ with edge costs ce_{ij}^1 , ce_{ij}^2 , f_{ij} and commodity set K , the survivable network design problem discussed in this thesis is to find a minimum cost subgraph of G such that between every pair in the commodity set K , there are at least 2-edge disjoint paths.

Cost ce_{ij}^1 is the cost of routing a unit flow on edge $\{i, j\} \in E$ on the primary path and cost ce_{ij}^2 is the cost of routing a unit flow on edge $\{i, j\} \in E$ on the secondary path. Cost f_{ij} is the fixed cost of activating edge $\{i, j\} \in E$. For each commodity $k \in K$, $s(k) \in V$ is the origin, $t(k) \in V$ is the destination, and $d(k)$ is the demand between the origin and destination. For quick reference to the notation used in this chapter the reader may look at Table 3.1.

G	: Given graph
V	: Vertex set in graph G
E	: Edge set in graph G
K	: Commodity set
ce_{ij}^1	: Unit routing cost of edge $\{i, j\} \in E$ for primary path
ce_{ij}^2	: Unit routing cost of edge $\{i, j\} \in E$ for secondary path
f_{ij}	: Fixed cost of activating edge $\{i, j\} \in E$
$s(k)$: Origin for commodity $k \in K$
$t(k)$: Destination for commodity $k \in K$
$d(k)$: Demand for commodity $k \in K$

Table 3.1: Notation

We use the following decision variables in our model:

xp_{ij}^k is the decision variable to keep track of which edge belongs to the primary path of the specific commodity at hand. Similarly, xs_{ij}^k is the decision variable which holds the edges belonging to the secondary path for each commodity. The last decision variable, y_{ij} is necessary to keep track of the edges that are –opened– in order to add the fixed costs of opening an edge to the objective function value.

$$xp_{ij}^k = \begin{cases} 1, & \text{if edge } \{i, j\} \in E \text{ is used in the direction from } i \text{ to } j \\ & \text{on the primary path of commodity } k \in K \\ 0, & \text{otherwise} \end{cases}$$

$$xs_{ij}^k = \begin{cases} 1, & \text{if edge } \{i, j\} \in E \text{ is used in the direction from } i \text{ to } j \\ & \text{on the secondary path of commodity } k \in K \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1, & \text{if edge } \{i, j\} \in E \text{ is used} \\ 0, & \text{otherwise} \end{cases}$$

Now, we can model SND_DP as follows:

$$\min \sum_{\{i,j\} \in E} f_{ij} y_{ij} + \sum_{k \in K} \sum_{\{i,j\} \in E} (ce_{ij}^1 d(k)(xp_{ij}^k + xp_{ji}^k) + ce_{ij}^2 d(k)(xs_{ij}^k + xs_{ji}^k)) \quad (3.1)$$

s.t.

$$\sum_{j:\{i,j\} \in E} xp_{ij}^k - \sum_{j:\{i,j\} \in E} xp_{ji}^k = \begin{cases} 1, & \text{if } i = s(k) \forall k \in K, \forall i \in V \\ -1, & \text{if } i = t(k) \forall k \in K, \forall i \in V \\ 0, & \text{otherwise} \end{cases} \quad (3.2)$$

$$\sum_{j:\{i,j\} \in E} xs_{ij}^k - \sum_{j:\{i,j\} \in E} xs_{ji}^k = \begin{cases} 1, & \text{if } i = s(k) \forall k \in K, \forall i \in V \\ -1, & \text{if } i = t(k) \forall k \in K, \forall i \in V \\ 0, & \text{otherwise} \end{cases} \quad (3.3)$$

$$xp_{ij}^k + xp_{ji}^k + xs_{ij}^k + xs_{ji}^k \leq y_{ij} \quad \forall k \in K, \forall \{i, j\} \in E \quad (3.4)$$

$$xp_{ij}^k, xs_{ij}^k \in \{0, 1\} \quad \forall k \in K, \forall \{i, j\} \in E \quad (3.5)$$

$$y_{ij} \in \{0, 1\} \quad \forall \{i, j\} \in E \quad (3.6)$$

Constraints (3.2) and (3.3) are flow balance constraints. Here since we are searching for two paths, for each commodity, two sets of flow balance equations are written; one for the primary path (3.2) and one for the secondary path (3.3).

However, finding two paths for each commodity is not adequate since the task is to find two –disjoint– paths. Therefore, an edge $\{i, j\} \in E$ can only be used in one path; primary or secondary path for each commodity. Constraint (3.4) satisfies this request by bounding the number of times a commodity can use an edge. This constraint also opens edge $\{i, j\} \in E$ if that edge has been used by a commodity in either its primary or secondary path.

The first part of the summation in the objective function; $\sum_{\{i,j\} \in E} f_{ij} y_{ij}$ is for

calculating the total costs incurred due to the activation of the edges. The second part of the summation; $\sum_{k \in K} \sum_{\{i,j\} \in E} (ce_{ij}^1 d(k)(xp_{ij}^k + xp_{ji}^k) + ce_{ij}^2 d(k)(xs_{ij}^k + xs_{ji}^k))$ is necessary for calculating the total routing costs. The demands are used in this calculation since, the variable costs ce_{ij}^1 and ce_{ij}^2 are costs for routing a unit demand.

As observed in the above given model, constraints (3.2) and (3.3) are flow balance constraints. Hence, if the edge set to be used is given, i.e., y_{ij} values are fixed, the problem boils down to finding 2-edge disjoint paths for each commodity. This implies that if the fixed cost f_{ij} values are small in value to routing costs ce_{ij}^1 and ce_{ij}^2 , the problem becomes easier. Furthermore, this problem is a Minimum Cost Network Flow (MCNF) problem if there were single edge costs c_{ij} for finding 2-edge disjoint paths. We refer to this problem as Single Commodity Routing Problem (SCRP). The idea of using MCNF is important since Suurballe's algorithm [12] takes its origins from this model. The detailed explanation for Suurballe's algorithm is presented in Chapter 4.

We define,

$$x_{ij}^k = \begin{cases} 1, & \text{if edge } \{i, j\} \in E \text{ is used in the direction from } i \text{ to } j \\ & \text{on a path for commodity } k \in K \\ 0, & \text{otherwise} \end{cases}$$

Now, we can model the SCRP as follows:

$$\min \sum_{k \in K} \sum_{\{i,j\} \in E} c_{ij} x_{ij}^k$$

s.t.

$$\sum_{j:\{i,j\} \in E} x_{ij}^k - \sum_{j:\{i,j\} \in E} x_{ji}^k = \begin{cases} 2, & \text{if } i = s(k) \forall k \in K, \forall i \in V \\ -2, & \text{if } i = t(k) \forall k \in K, \forall i \in V \\ 0, & \text{otherwise} \end{cases}$$

$$0 \leq x_{ij}^k \leq 1 \quad \forall k \in K, \forall \{i, j\} \in E$$

The number of commodities used in SND-DP severely enlarges the problem in terms of the memory that is used. As the number of nodes increases, so does the number of commodities. This is because by assumption, in this thesis all of the possible combinations of source and destination pairs exist. In addition, the possible density of the graph; the amount of edges that are available, also effects the memory usage.

Differing edge costs for primary and secondary paths are other factors that make the problem harder (Recall that Suurballe and Tarjan's algorithm solves to optimality the 2 edge-disjoint problem with single routing costs for edges for a single commodity [12]). Throughout this thesis the assumption is that the relationship between the routing costs is: $1/2ce_{ij}^1 = ce_{ij}^2$.

In the next chapter, different construction and improvement heuristics are expressed.

Chapter 4

Heuristic Algorithms

In our problem, we find 2-edge disjoint paths between source and destination pairs. The objective is to minimize the total costs. These costs include primary and secondary path routing costs and fixed costs for opening edges. Recall that primary path costs is 2 times the secondary path costs.

To solve our problem, in this chapter, we make use of several algorithms. Firstly, Suurballe's algorithm is described since it is one of the basic algorithms utilized in the heuristic algorithms that are illustrated in this section. Afterwards two of the construction heuristics, namely, the two-step and one-step algorithm are described. These algorithms find initial feasible solutions for our problem. Finally, before concluding this chapter we explain improvement methods that can be applied to both of the construction heuristics. The improvement methods are referred to as the IP based heuristic, edge deletion heuristic, edge addition heuristic and cycle algorithm. These methods are explained in depth in section Improvement Heuristics.

4.1 Suurballe's Algorithm

Given a graph $G = (V, E)$, with single edge costs c_{ij} for each edge $\{i, j\} \in E$, a source node s , and destination node d , the survivable network design problem solved by Suurballe finds 2-edge disjoint paths between s and d while minimizing the total cost of these 2 paths. Cost c_{ij} is the cost of routing a unit flow on edge $\{i, j\} \in E$.

Suurballe and Tarjan [12] describe Suurballe's algorithm that solves the 2-edge disjoint paths problem. The method is based on the generic algorithm explained in [13] which is for finding node-disjoint paths. The altered version of the algorithm, which solves the 2-edge disjoint paths problem, runs in $O(m \log_{1+m/n} n)$ time and $O(m)$ space, where m is the number of edges and n is the number of nodes given in a graph G .

The algorithm starts by finding a shortest path tree from node s to all other nodes using Dijkstra's algorithm [1]. Afterwards, the original graph G is altered by changing the cost values of the edges, while everything else remains the same. The new edge costs for an edge $\{u, v\}$ are calculated as follows:

$\pi_{u,v} = c_{u,v} + d_{s,u} - d_{s,v}$, where $c_{u,v}$ represents the original cost values of the edges, $d_{s,u}$ is the shortest path distance from node s to u and $d_{s,v}$ is the shortest path distance from node s to v . The new edge costs are simply the reduced costs from LP duality when the LP model in discussion is the relaxation of SCRP. Then, the edges' directions used in the shortest path from node s to node d are reversed. The shortest path from node s to node d is calculated again using new edge costs and new edge directions for the edges found on the previous shortest path. After removing the links that appear (in opposite direction) in both the original shortest $s - d$ path and the latter shortest path $s - d$ tree 2-link disjoint paths between nodes s and d can be easily constructed. The pseudo-code of the algorithm can be found in Algorithm 1.

For a deeper understanding of the algorithm one can analyze the example in Figure 4.2. In this example 2-edge disjoint paths from node 0 to node 5 are to be found. Numbers next to edges which are inside rectangles represent edge

Algorithm 1 Suurballe

Compute the shortest-path tree rooted at node s using Dijkstra's algorithm [1].

Let $d_{s,u}$ denote the shortest-path distance from node s to node u .

Transform the original graph G to an auxiliary graph G' as follows:

Node and links are kept unchanged.

The cost of each link $\{u, v\}$ in G' is defined by

$\pi_{u,v} = c_{u,v} + d_{s,u} - d_{s,v}$, where $\pi_{u,v}$ denotes the cost of link $\{u, v\}$ in graph G' and $c_{u,v}$ denotes the cost of link $\{u, v\}$ in graph G .

Reverse the directions of the links along the shortest path from node s to node d .

Compute the shortest path from node s to node d in graph G' .

The shortest path between nodes s and d in $G(G')$ is denoted as $P(P')$.

Remove the links appearing in both P and P'

(in opposite direction), all the other links in P and P'

form a cycle when ignoring their directions. Two link-disjoint paths between nodes s and d are found from the cycle.

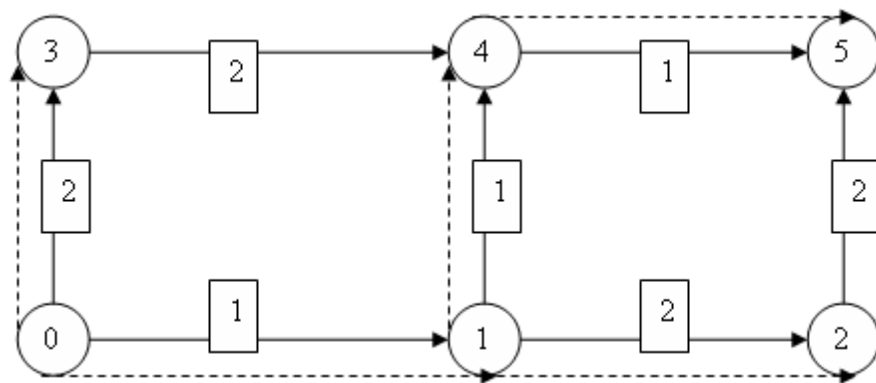
costs. The dashed lines show the shortest-path tree rooted at node 0. The shortest path from node 0 to node 5 is $P = 0 \rightarrow 1 \rightarrow 4 \rightarrow 5$. After completing the transformation of the graph from G to G' , the shortest path between nodes 0 and 5 is $P' = 0 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 5$, which can be observed in part (b) of the figure. After removing the arcs $(1, 4)$ and $(4, 1)$, the 2-edge disjoint paths $0 \rightarrow 3 \rightarrow 4 \rightarrow 5$ and $0 \rightarrow 1 \rightarrow 2 \rightarrow 5$ are found and shown in part (c) of the figure.

4.2 Construction Heuristics

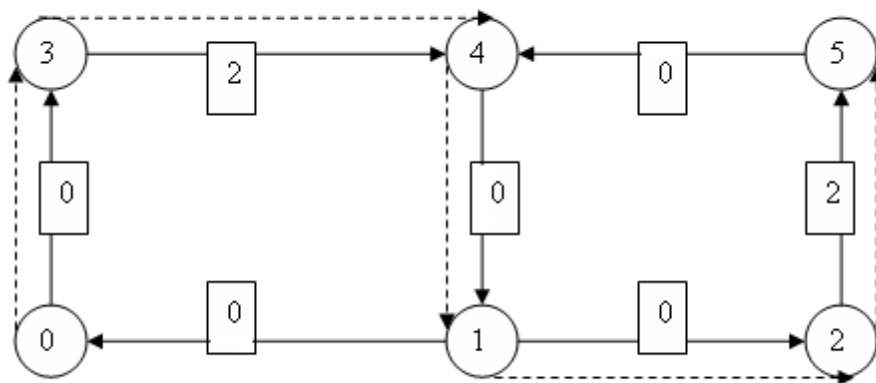
In this section, the construction heuristics that utilize both Dijkstra's and Suurballe's algorithm are described. These heuristics find initial feasible solutions.

4.2.1 Two-Step Algorithm

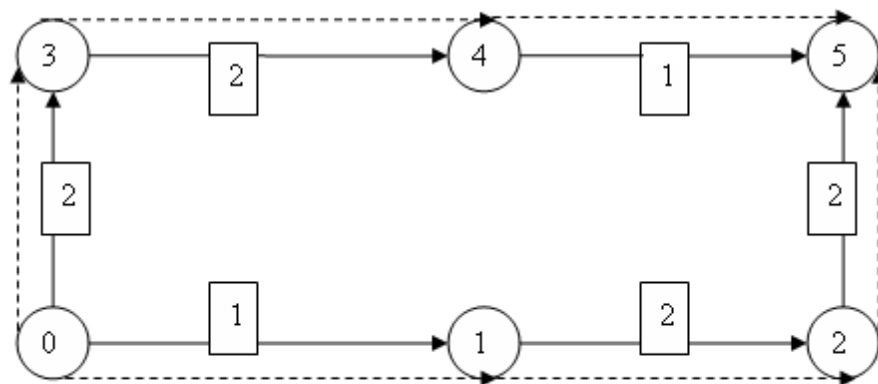
The basic idea used in the two-step algorithm is the application of Dijkstra's algorithm [1] using single link costs that are equal to primary path costs, ce_{ij}^1 . For



(a)



(b)



(c)

Figure 4.1: An example which shows how Suurballe's algorithm works

each commodity Dijkstra's shortest path algorithm is applied once to construct the primary path. Then, the edges used in the primary path are temporarily deleted from the network so that the second application of the Dijkstra's algorithm does not use these edges. By this way, the secondary path is also found assuming 2-edge disjoint paths exist for that commodity. This process is done as if only the particular commodity at hand exists, i.e., the total costs for each commodity are calculated separately without considering active or inactive edges. Thus, the fixed costs are not included in the total costs. Afterwards the commodity in discussion is placed in a commodities array which holds the total costs (primary path cost + secondary path cost) for that commodity. Finally, when the total cost calculation for each commodity is completed the values in this array are sorted according to descending order of total costs for each commodity.

At the end of the above process the problem is solved again according to the order of commodities in the commodities array. However, in this case the fixed costs are also considered while finding total costs for each commodity. Furthermore, each commodity is not thought of separately, i.e., after a commodity's two paths are fixed and total costs are calculated the next commodity does not pay any fixed costs if it uses edges that were previously activated. Until all of the commodities in the commodities array are processed the assignment of paths continues. At the end a feasible solution is obtained.

To improve the feasible solution at hand, a "rerouting" procedure is also performed. In this procedure, only the edges which were activated during the above explained steps are used and the fixed costs are added to the total costs at the end of the algorithm. Basically, for every commodity two paths are found again assuming that only the active edge set provided by the above steps is available. Furthermore, if any of the edges which were activated as a result of the above process are not used during the rerouting procedure, they are left out of the active edge set and their fixed cost values are not included in total costs.

The important point to consider in this algorithm is the order of connection of the commodities. In other words, the key question is which commodity should be processed first since the primary and secondary paths constructed for that

commodity become permanent. This is essential in terms of activating the edges. If the fixed cost values are large in value compared to routing costs ce_{ij}^1 and ce_{ij}^2 the order of activating edges becomes crucial in finding good solutions; solutions that are closer in value to the optimal solution.

A number of approaches have been considered in determining the order of commodities to be processed. One of these approaches calculates the total costs (objective value) for each commodity as if other commodities do not exist by using the two-step algorithm. However, during this calculation fixed costs are not included in the total costs; only routing costs are considered. Afterwards, the commodities are listed in descending values in terms of their total costs (This process is explained in the first paragraph). Hence, the commodity that gives the largest total cost is processed first by this approach and the commodity that gives the smallest total cost is processed last. Furthermore, this approach can also be applied multiple times by using a dynamic calculation technique. After processing the first commodity in the list, several edges are activated. By using this information the calculation of the total costs for each commodity can be made again and the new ordering will probably be different from the previous ordering since the activated edge set will be different after the first(in the list) commodity's paths are made permanent. This dynamic calculation technique can be repeated as many times as $|K| - 1$ where $|K|$ is the number of commodities. However, although the results provided by this dynamic calculation technique may be better compared to ordering the commodity list only once, the running times of the dynamic calculations will be much higher. Another approach is to randomly list the commodities. Unfortunately, the drawback of this approach is that the total cost found by using random listing can be very close to the optimal solution in some cases and in others far away from the optimal solution. The unstability of random listing makes this approach unfavorable.

After testing these approaches, we have decided to use the static listing of commodities. Calculating the total costs once for each commodity has better times compared to dynamic listing. Furthermore, several improvement methods have been discussed that will improve the quality of the solution and still obtain considerably less running times. As discussed above, the random listing approach

has not been chosen due to its unstable results. The pseudo-code for the two-step algorithm that lists the commodities according to static listing and by following this list processes the commodities one by one can be found in Algorithm 2.

Algorithm 2 Two-Step

for all $k \in K$ **do**

Using Dijkstra's algorithm (edge costs according to ce_{ij}^1) calculate the shortest(primary) path from source $s(k)$ to destination $t(k)$

Temporarily delete the edges used on the primary path for commodity k from the given graph G

Using Dijkstra's algorithm (edge costs according to ce_{ij}^2) calculate the shortest(secondary) path from source $s(k)$ to destination $t(k)$

Place commodity k in *commoditiesArray* after calculating the total costs as primary path cost + secondary path cost

Sort the *commoditiesArray* in descending order of total costs

for all $k \in \textit{commoditiesArray}$ **do**

Using Dijkstra's algorithm calculate the shortest(primary) path from source $s(k)$ to destination $t(k)$ using the following costs for edges

if Edge has been activated before **then**

Cost of edge = ce_{ij}^1

else if Edge has not been activated before **then**

Cost of edge = $ce_{ij}^1 + f_{ij}$ and activate edge $e = \{i, j\}$

Temporarily delete the edges used on the primary path for commodity k from the given graph G

Using Dijkstra's algorithm calculate the shortest(secondary) path from source $s(k)$ to destination $t(k)$

if Edge has been activated before **then**

Cost of edge = ce_{ij}^2

else if Edge has not been activated before **then**

Cost of edge = $ce_{ij}^2 + f_{ij}$ and activate edge $\{i, j\}$

Reroute and close unused edges

4.2.2 One-Step Algorithm

In one-step algorithm instead of the naive approach of merely using Dijkstra's algorithm, Suurballe's algorithm is utilized. The underlying process used in both algorithms; the one-step and two-step algorithms are the same. Both of the algorithms initially find the total costs for each commodity as if other commodities

do not exist in the problem and a listing of the commodities is done according to descending order of total costs. The total costs include the routing costs; fixed cost values are not considered. Thus, the basic difference is the usage of Suurballe's algorithm for one-step algorithm. At the very end of the one-step algorithm, a "rerouting" procedure is performed like the rerouting procedure explained in the two-step algorithm.

In addition to the importance of listing the commodities as explained previously, one of the other crucial factors in using Suurballe's algorithm is to determine which costs will be used during the utilization of the algorithm. This is important since Suurballe's algorithm assumes that only a single cost for an edge is available.

For the calculation of the initial total costs, the edge costs are arranged as primary path costs; ce_{ij}^1 for the Suurballe's algorithm. However, after the sorting of the commodities according to descending order of total costs is completed the former approach is altered. During the second application of Suurballe's algorithm for each commodity, the edge costs remain as primary path costs (ce_{ij}^1) if that specific edge has not been activated before else, the cost of the edge becomes primary path costs (ce_{ij}^1) + fixed cost of that edge. Since by using the one-step algorithm we make the fixed cost (f_{ij}) values the important factors in defining the paths, the solution tries to choose all of the edges with lower fixed costs (f_{ij}) if available. This makes the problem favor some edges over others. By altering the edge costs the favoring of the edges can become more balanced. This can be achieved by considering the weight (w_{ij}) of an edge. The one-step algorithm is altered to create the one-step version 2 algorithm by changing the edge costs. Instead of checking if an edge has been activated and arranging the costs to include the fixed costs if that edge has not been activated before a more complex method is used. In this method, during the second application of the Suurballe's algorithm for each commodity the edge costs are arranged as primary path costs (ce_{ij}^1) + fixed cost (f_{ij}) / weight of edge (w_{ij}). The edge weights (w_{ij}) are calculated during the first application of Suurballe's algorithm in the part before the commodities are sorted according to their total costs in the one-step algorithm. The edge weights (w_{ij}) for a particular edge $\{i, j\}$ is equal

to the amount of routing done throughout that edge. The routing amount for an edge is calculated by considering –how much– demand that edge carries. By cumulatively adding all of the demands that are carried by edge $\{i, j\}$ we obtain the weight of edge $\{i, j\}$, w_{ij} . In compact form, the weight of an edge, w_{ij} is equal to $\sum_{k \in K: k \text{ uses edge on primary path}\{i,j\}} d_k + \sum_{k \in K: k \text{ uses edge on secondary path}\{i,j\}} 1/2d_k$.

Furthermore, the outcome of Suurballe’s algorithm is two paths. The decision of making which path to be primary and which path to be secondary is important in reducing the total costs as much as possible. Both of the alternatives are considered by assigning one path to be primary path and other to be secondary path and vice versa. The assignment which provides the lowest total cost is chosen and the activation of the edges are done accordingly. The pseudo-code for the one-step algorithm that processes the commodities according to static listing one by one can be found in Algorithm 3.

Algorithm 3 One-Step

for all $k \in K$ **do**

Using Suurballe’s algorithm (edge costs according to ce_{ij}^1) calculate two paths from source $s(k)$ to destination $t(k)$

Make the path with the larger total cost value the secondary path and the other primary path for commodity k

Place commodity k in *commoditiesArray* after calculating the total costs as primary path cost + secondary path cost

Sort the *commoditiesArray* in descending order of total costs

for all $k \in \textit{commoditiesArray}$ **do**

Using Suurballe’s algorithm calculate two paths from source $s(k)$ to destination $t(k)$

if Edge has been activated before **then**

Cost of edge = ce_{ij}^1

else if Edge has not been activated before **then**

Cost of edge = $ce_{ij}^1 + f_{ij}$ and activate edge $\{i, j\}$

Assign one path to be primary path and other to be secondary path and vice versa.

Pick the arrangement with the lowest costs and change the activated edge information if necessary

Reroute and close unused edges

From this point on there will be no discussion about the two-step algorithm,

this is because the computational results for the two-step algorithm are very poor when compared to one-step algorithm's results. This is an implied result since the two-step algorithm eliminates all of the edges that are used on a primary path from a source to a destination in order to find a disjoint secondary path for the same pair. However, one-step algorithm utilizes Suurballe's algorithm and in this algorithm the edges used on a primary path for a source and destination pair are not completely removed from the graph before calculating the secondary paths for the same pair. Instead, the edges utilized on the primary path's directions(orientations) are reversed. Hence, merely using two applications of Dijkstra's algorithm provides a smaller subset of edges for the search of secondary paths compared to Suurballe's algorithm.

4.3 Improvement Heuristics

In this section several improvement methods for the one-step and one-step version 2 algorithms are explained (These improvement methods can also be applied to the two-step algorithm but no computational results are presented for the two-step algorithm due to the explanation made in the previous paragraph). One-step and one-step version 2 algorithms are constructive heuristics and essentially provide initial feasible solutions that can be further improved. All of the improvement methods can be applied to both of the algorithms.

4.3.1 IP Based Heuristic

IP based heuristic basically does what "rerouting" procedure does but it finds the optimal solution for the resulting active edge set that is found after the application of one of the construction heuristics. In other words, the active edge set is provided to the model described in Chapter 3 and the fixed costs are omitted from the objective function. After an optimal solution to this problem is found the edge set is checked for any edge that may have become inactive and only then the fixed cost values are calculated for the new active edge set and added to the



Figure 4.2: Steps of Edge Deletion Heuristic

problem. Although IP based heuristic may provide better solutions compared to the “rerouting” procedure, the resource usage of IP based heuristic is much higher.

4.3.2 Edge Deletion

In this improvement all of the active edges provided by one of the construction heuristics are made inactive one by one and the problem is solved again for the same algorithm. After making an edge inactive the problem may become infeasible hence, in these situations the edge is reactivated without checking the total cost value. Furthermore, making an edge inactive may increase the previous total costs so, inactivating that edge is not favorable. An edge is made inactive only when the new solution to the problem with the new active edge set has smaller total costs compared to the previous costs and it provides a feasible solution. This improvement method can also be applied using GAMS with the model in Chapter 3. However, having to do as many iterations as the active edges provided by one of the algorithms can be very costly in terms of the running times. The reader can view Figure 4.2 for the steps of Edge Deletion Heuristic.

4.3.3 Edge Addition

Edge addition is the reverse of edge deletion. Inactive edges; edges that are not provided by the result of one of the construction heuristics are made active one by one. There is no infeasibility in this case since edges that were inactive are made active (the active edge set becomes larger) and the problem is solved again with



Figure 4.3: Steps of Edge Addition Heuristic

the algorithm that had provided the inactive edges. If the total costs decrease, a new solution is found. Total costs can decrease in value if some other previously active edge has become inactive. However, if the total costs increase, the edge that was activated is inactivated once again. This process continues until all of the inactive edges have been activated once. The reader can view Figure 4.3 for the steps of Edge Addition Heuristic.

4.3.4 Cycle Algorithm

Having tested improvement heuristics that merely add or delete edges but do not combine both approaches we thought that we could unite these heuristics under the same algorithm. Therefore, in this section we describe the cycle algorithm that both removes and adds edges to a solution at a single iteration. The algorithm utilizes the one-step version 2 algorithm. Essentially an alternative cycle after obtaining a feasible solution for a source-destination pair is sought for. The steps of the cycle algorithm for a single source and destination pair; i and j can be observed in Figure 4.4. In part (a) edge $\{i, j\}$ shows the primary path and the dashed curved line shows the secondary path for the i and j pair. This solution is obtained via the usage of one-step version 2 algorithm. In part (b), if possible, a third path is found such that no common edges between this path and primary and secondary paths exist. After closing edge $\{i, j\}$, we still have two paths for the i and j pair, as shown in part (c).

The algorithm starts by finding an initial feasible solution by applying the one-step version 2 algorithm. Afterwards, for every edge $\{i, j\}$ in the activated edge set, E' obtained via the initial feasible solution, edges in the primary and secondary paths from source i to j is removed. Dijkstra's shortest path algorithm

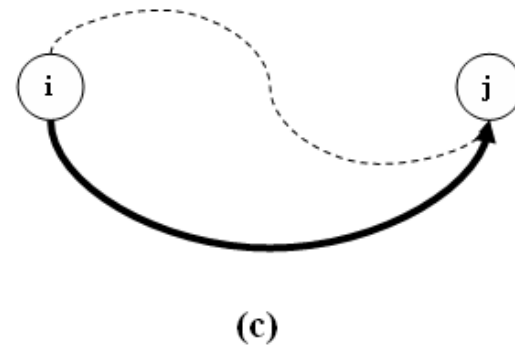
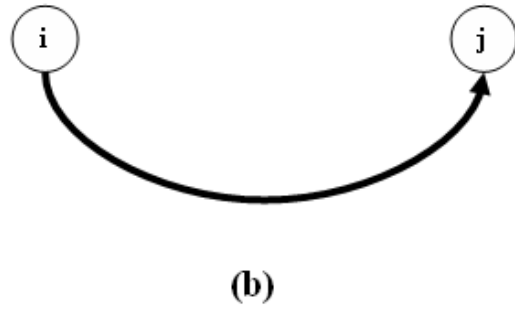
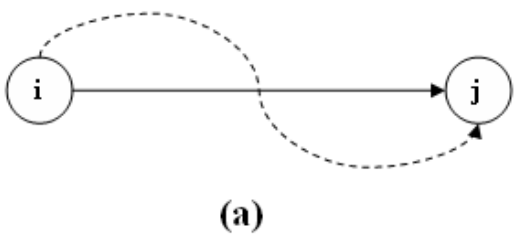


Figure 4.4: An example for a single commodity i and j which shows how cycle algorithm works

is applied to the i and j pair. The edge costs are arranged according to the activated edge set E' , if an edge belongs to the edge set obtained from the initial feasible solution (E') its cost is equal to primary path costs, ce_{kl}^1 , if an edge belongs to the edge set of the original graph but has not been activated in the initial feasible solution ($E - E'$) its cost is equal to primary path costs (ce_{kl}^1) + fixed cost (f_{ij}) / weight of edge (w_{ij}) (w_{ij} is described in the previous section). If a new shortest path from i to j is not found (no path exists) the algorithm continues with the next edge in the activated edge set, E' else, edge $\{i, j\}$ is inactivated and –rerouting– described in the two-step algorithm is applied once again. The cost arrangement for the edges is the same as the arrangement used for applying Dijkstra’s algorithm. If closing edge $\{i, j\}$ provides a new feasible solution with improved costs then the edge stays inactive else the edge is reactivated. This process continues till there are no more edges to process in the initial activated edge set provided by the initial feasible solution. The pseudo-code for the cycle algorithm can be found in Algorithm 4.

Algorithm 4 Cycle

```

Run One-step version 2 algorithm
current cost is equal to the total cost returned by the algorithm
for all  $e = \{i, j\} \in E'$  do
  Temporarily delete the edges used in primary and secondary paths for
   $i$ (source) and  $j$ (destination) pair
  Find a new shortest path from  $i$  to  $j$  using Dijkstra’s algorithm
  Using edge costs
  if Edge  $e = \{k, l\} \in E'$  then
    Cost of edge  $e = ce_{kl}^1$ 
  else if Edge  $e = \{k, l\} \in E - E'$  then
    Cost of edge  $e = ce_{kl}^1 + f_{ij}/w_{ij}$ 
  if a shortest path from  $i$  to  $j$  is found then
    inactivate  $e = \{i, j\}$ 
    Reroute using the same edge cost structure used for running Dijkstra’s
    algorithm
    if new cost returned from rerouting < current cost then
      current cost = new cost
    else if new cost returned from rerouting > current cost then
      activate  $e = \{i, j\}$ 

```

Having completed describing the construction algorithms and improvement

methods in this section, in Chapter 5 we take a look at the computational results provided by these algorithms and improvement methods.

Chapter 5

Numerical Results of Algorithms

In this chapter the algorithms and improvement methods described in Chapter 4 are tested on a computer with 2.6 GHz AMD Opteron 252 processor and 2 GB of RAM operating under the system CentOS (Linux version 2.6.9-42.0.3.ELsmp). Furthermore, in order to obtain the optimal solution values for the test instances, we solved the IP model presented in Chapter 3 by using GAMS 22.5 and CPLEX 11.0.0 on the same computer.

5.1 Test Instances

In this section the characteristics of the test instances are explained. The running times of the algorithms and improvement methods are effected differently according to particular aspects of the test instances. Increasing the node number and edge numbers also increase the running times. In addition, the memory usage increases and in some test instances the computer runs out of memory. Furthermore, relationship between routing costs ce_{ij}^1 , ce_{ij}^2 and fixed cost f_{ij} severely effect the running times of the IP model. If the routing costs and fixed costs are in proximity of each other, the problem becomes easier so the running times decrease. However, if the routing costs and fixed costs are very different from each other and the fixed cost values are extremely higher than routing costs then

the running times severely increase. Therefore, a bound of 60 minutes is used for any test case and any method of solving the problem in this thesis.

5.1.1 Node Number (V)

The number of nodes selected for the test instances is 30 and 40. This choice is due to the memory restrictions of the mathematical model. Comparison of the optimal solution provided by the mathematical model and the total cost value obtained from the algorithms along with their improvements is impossible for larger instances when an optimal solution cannot be found. In these cases the lower bound provided by the IP model is used for comparison purposes.

5.1.2 Edge Number (E)

Edges are generated randomly according to three different density levels; 0.5, 0.75 and 1 respectively. Each corresponds to the probability of an edge appearing in the graph.

5.1.3 Primary and Secondary Path Costs (ce_{ij}^1 and ce_{ij}^2)

Nodes are randomly selected from a 100×100 grid and the edge distances are simply calculated as the Euclidean distances between the points. This process is done for each edge of the network. The euclidean distance found is set as the secondary path costs (ce_{ij}^2) of each corresponding edge. To obtain the primary path costs, $ce_{ij}^1 = 2ce_{ij}^2$ calculations are done for each edge of the network.

5.1.4 Fixed Cost (f_{ij})

There are three components that make up a fixed cost; a random number, a coefficient c and primary path costs (ce_{ij}^1).

fixed cost f_{ij} for some edge $\{i, j\} = \text{RandomNumber} + c \times ce_{ij}^1$ (primary path cost of some edge (i, j))

The random number $\in [0, 1000]$. Coefficient c is either one of 1, 10 or 100 in different settings. Here the random number is assigned according to the geographical conditions the fibers are installed. For example, having to install fibers underground and aboveground have different costs. The coefficient c is a parameter for us to detect how the IP model and the algorithms behave when the range between fixed costs f_{ij} and routing costs ce_{ij}^1 vary.

5.1.5 Demand (d)

Every possible combination of commodities is available according to the node number 30 or 40. However, a commodity with source node s and destination node d , and source node d and destination node s use the same primary and secondary paths. Hence, adding both demands and finding 2-edge disjoint paths from source node s to destination node d is adequate. Two demand value ranges are possible; either $d \in [1, 10]$ or $d \in [1, 100]$.

5.2 Results of Test Instances

In this section the results of the one-step and one-step version 2 algorithm are presented. Furthermore, a comparison between the edge deletion improvement heuristic and cycle algorithm is done. For detailed tables the reader may look at the Appendix. In the Appendix the running times along with the number of edges that were activated in the relevant algorithms are also presented.

The IP based heuristic results for the one-step algorithm have been tested. However, the running times are slow therefore, the “rerouting” procedure described in the construction heuristics is utilized instead. This improvement heuristic has not been used to test one-step version 2 algorithm due to its running times. The interested reader may take a look at the detailed results in the Appendix.

No graphs showing the results of the edge addition heuristic are presented in this section. This is because, the edge addition heuristic has not improved the objective value for the one-step version 2 algorithm for both cases, when $d \in [1, 10]$ and $d \in [1, 100]$. In other words, after applying edge addition heuristic to the result of the one-step version 2 algorithm none of the unused edges became active. However, the reader may observe the detailed results in the Appendix.

5.2.1 One-step and One-step-2 Algorithm Comparison

In this section the results of the test instances for comparing the one-step and one-step-2 algorithms is presented. The graphs show the %gap which is calculated according to the optimal solution in the y-axis and the three levels of density of the graphs in the x-axis. Specifically, $\%gap = (\text{heuristic value} - \text{optimal value}) / \text{optimal value} \times 100$. For each point on the graph that coincide with the density levels and for each algorithm, 3 sample instance results' %gaps are averaged. As mentioned earlier, the reader can observe each test instance in detail in the Appendix section.

Figure 5.1 show the results of the test instances for node $\# = 30$ and for each coefficient value c ; 1, 10 and 100. Additionally we assume that $d \in [1, 100]$ for the graphs aligned to the left and we assume that $d \in [1, 10]$ for the graphs aligned to the right. It is observed from the figure that the increase in the coefficient value increases the %gap. The largest %gap is observed in the last graph of the figure where $c = 100$ for both demand value ranges. This is also true for the test instances when node $\# = 40$ (see Figure 5.2). We can also compare the results obtained from the two different demand ranges where $d \in [1, 100]$ and $d \in [1, 10]$. The %gap values are considerably high when $d \in [1, 10]$ for both node $\# = 30$ and node $\# = 40$ cases, this is because lowering the demand values makes the same effect as increasing the fixed cost values (increasing the c value) since we try to minimize total costs for each commodity and each commodity carries demand that the total cost includes by multiplying the demand values with the routing costs.

When $c = 1$ for both cases where node $\# = 30$ and node $\# = 40$ the %gap of both algorithms are nearly identical for $d \in [1, 100]$. However, when $c = 10$ and node $\# = 30$, one-step algorithm provides lower %gap values but when $c = 10$ and node $\# = 40$, one-step-2 algorithm provides lower %gap values again for $d \in [1, 100]$. In the final coefficient value, that is when $c = 100$ for both cases where node $\# = 30$ and node $\# = 40$ the %gap of one-step-2 algorithm is better for both $d \in [1, 100]$ and $d \in [1, 10]$. Actually, when $d \in [1, 10]$ in all of the instances one-step-2 algorithm provides better %gap values. This result can be predicted since the one-step algorithm favors edges (as explained in Chapter 4), and in all our test cases activates a smaller set of edges compared to the one-step-2 algorithm. The small set of edges cannot provide good results in some cases, especially when the difference between the fixed cost values and the routing costs severely increase, that is when $c = 100$.

5.2.2 Edge Deletion Heuristic and Cycle Algorithm Comparison

The same instances that were used to compare one-step and one-step-2 algorithms are also used in this section to compare the edge deletion heuristic and the cycle algorithm's performances. In these instances only the one-step-2 algorithm is utilized for the improvement methods this is because one-step-2 algorithm produces better results in terms of the %gap, this can be observed in Figures 5.3 and 5.4 where the construction heuristic is the one-step algorithm for the graphs aligned to the right and the construction heuristic is the one-step-2 algorithm for the graphs aligned to the left and also $d \in [1, 100]$ (Especially when the c values get larger the difference can be observed easily). From this point on, we only consider the one-step-2 algorithm as the best construction heuristic method and utilize it in our improvement heuristics. This result is due to one-step algorithm's edge favoring aspect. The improvement heuristics show better performances for the one-step-2 algorithm since the active edge set provided by the one-step algorithm is, in all our test cases, smaller than the active edge set provided by the one-step-2 algorithm's. This is due to the fact that when there are a larger number of edges

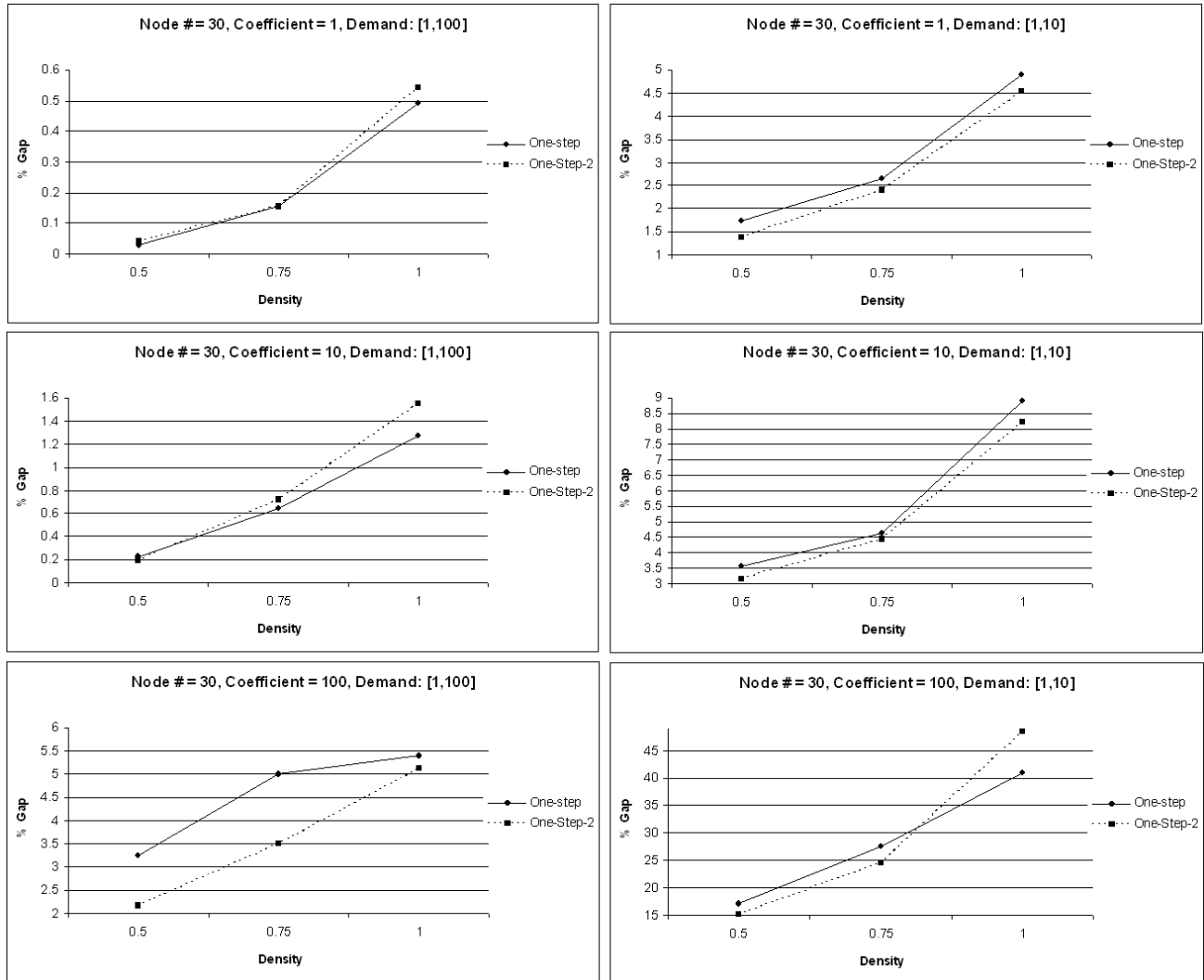


Figure 5.1: One-step and One-step-2 Algorithm Results for node # = 30

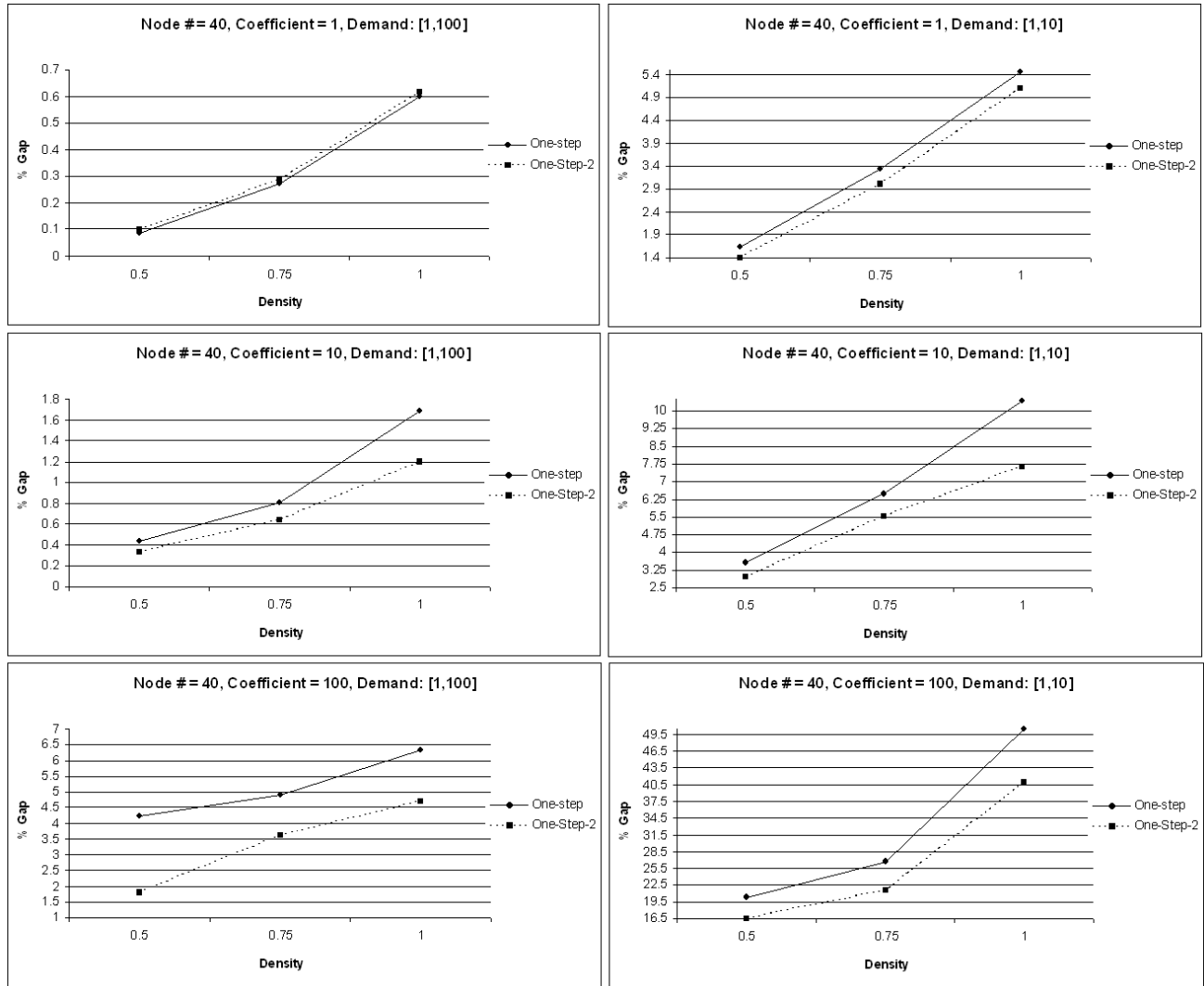


Figure 5.2: One-step and One-step-2 Algorithm Results for node # = 40

to select from a better solution can be obtained.

Similar to the above section, the graphs show the %gap which is calculated according to the optimal solution in the y-axis and the three levels of density of the graphs in the x-axis. For each point on the graph that coincide with the density levels and for each algorithm, 3 sample instance results' %gaps are averaged.

In both Figures 5.5 and 5.6 the cycle algorithm beats the edge deletion heuristic in terms of the %gap or provides nearly identical results in a small number of cases. This result can be predicted since cycle algorithm includes all of the processes utilized in the edge deletion heuristic. The demand ranges, $d \in [1, 100]$ and $d \in [1, 10]$ do not change the relationship between the edge deletion heuristic and cycle algorithm, for every graph in Figures 5.5 and 5.6 cycle algorithm shows better results for the %gap values. But, for $d \in [1, 10]$, our results are far worse compared to $d \in [1, 100]$ case. The reason for this has been explained in the previous section. However, the reader may notice that although the %gap are considerably large for the one-step-2 algorithm in Figures 5.1 and 5.2 (for $d \in [1, 10]$) when the improvement heuristics are run the %gap's decrease by a large amount.

5.2.3 Summary of Our Findings

For the convenience of the reader, in this section we incorporate some of the graphs presented in the above figures and summarize our findings.

In Figure 5.7 we have unified the graphs for the coefficient value c for node $\# = 30$ and $\# = 40$ to compare the one-step and one-step-2 algorithm. From these graphs it can be visualized more clearly how the %gap's increase when the c values increase. Also the range of the %gap values is very large when $d \in [1, 10]$ for both node ($\# = 30, 40$) cases. Almost in all the test instances one-step-2 algorithm presents better %gap values or very similar results when compared to the one-step algorithm.

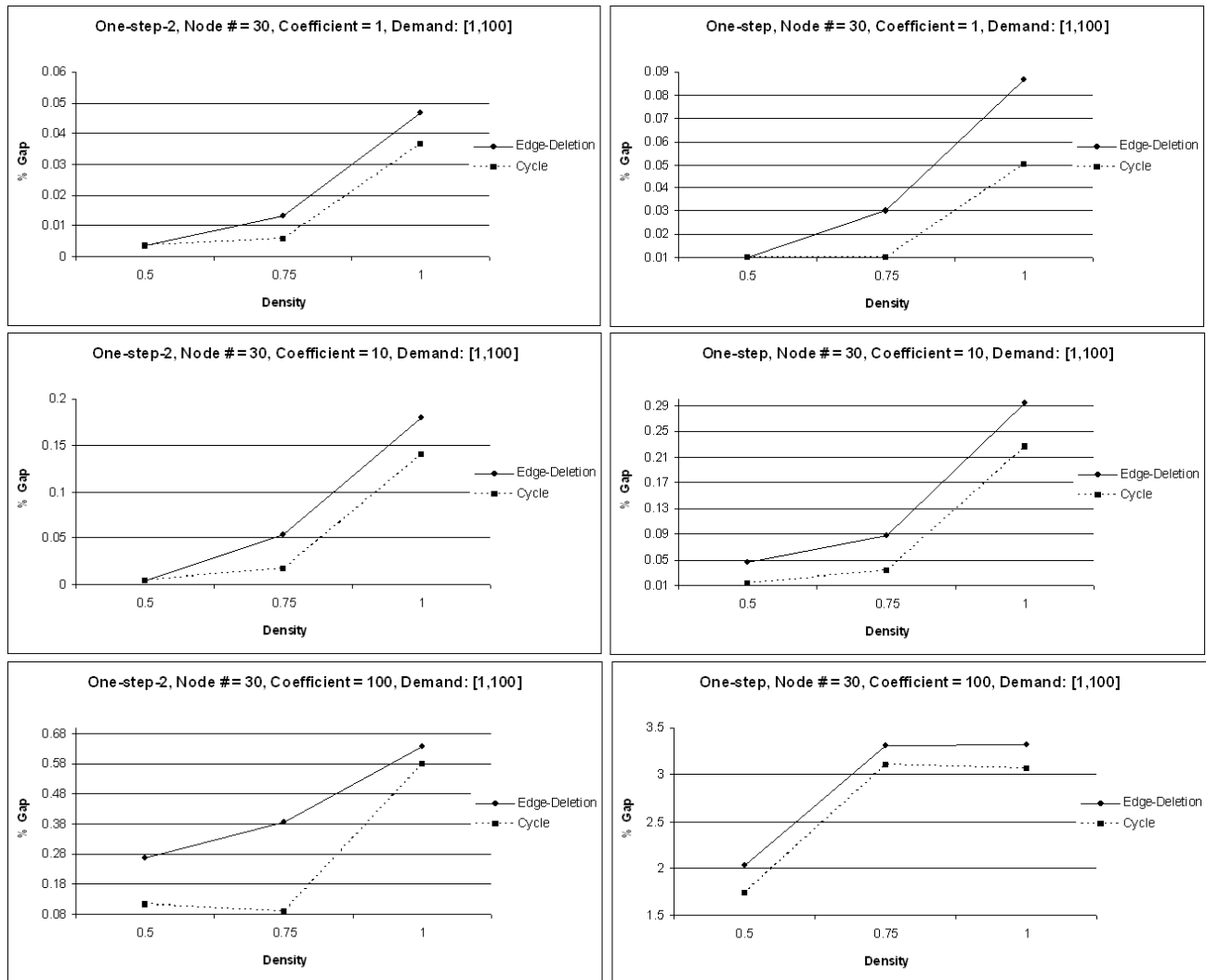


Figure 5.3: Edge Deletion Heuristic and Cycle Algorithm Results for node # = 30 and Comparison Results for One-step and One-step-2 Algorithms as the Construction Heuristics

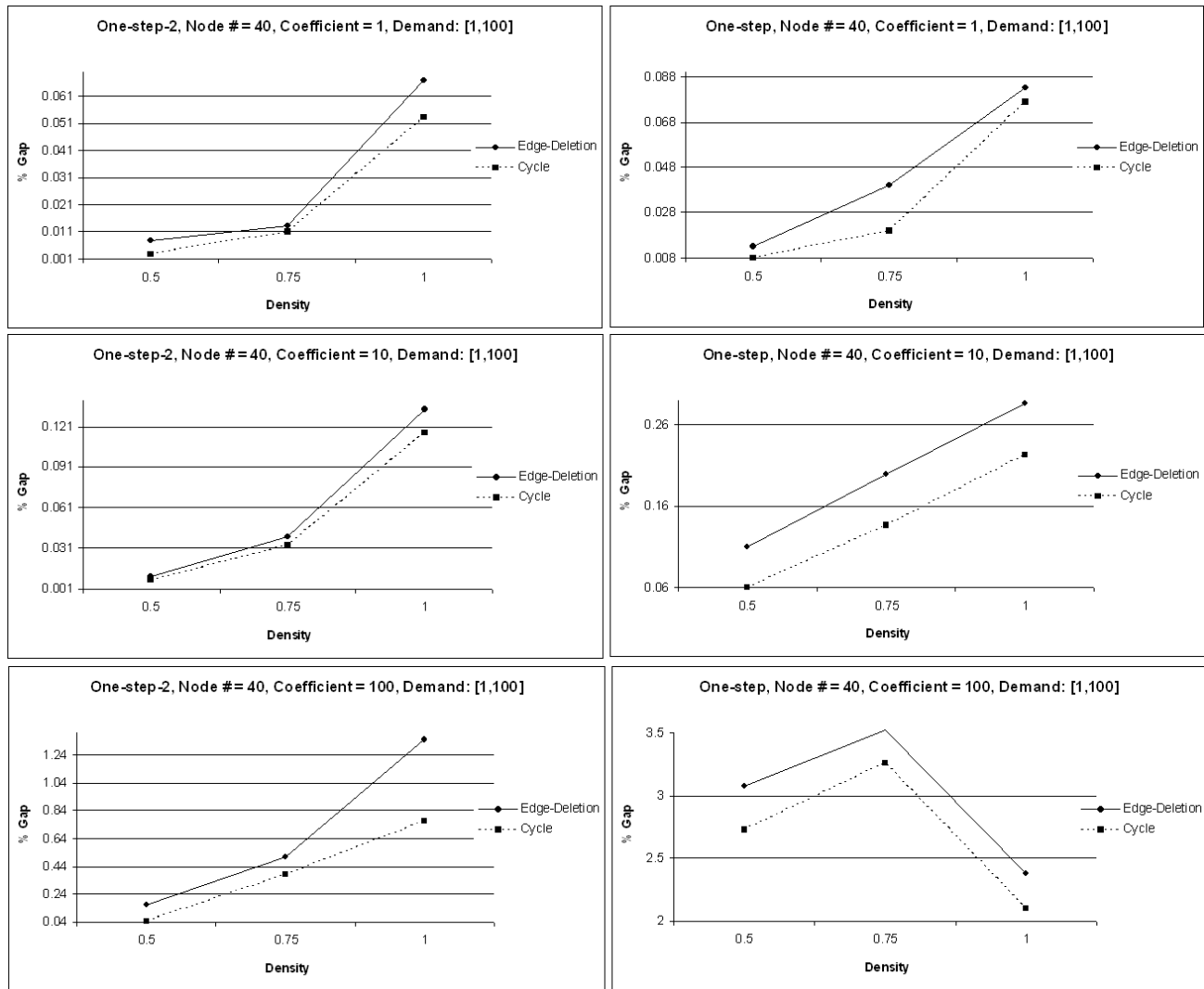


Figure 5.4: Edge Deletion Heuristic and Cycle Algorithm Results for node # = 40 and Comparison Results for One-step and One-step-2 Algorithms as the Construction Heuristics

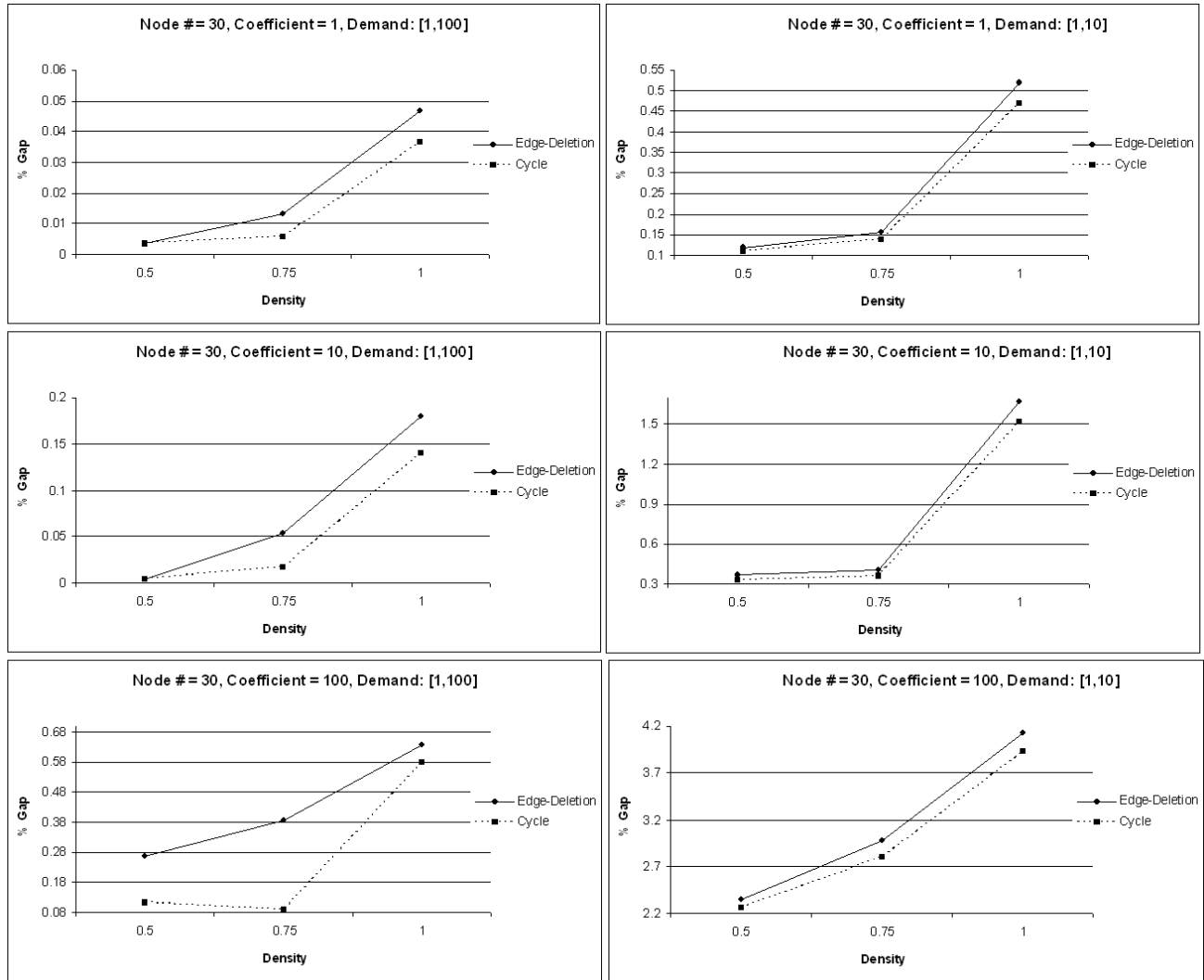


Figure 5.5: Edge Deletion Heuristic and Cycle Algorithm Results for node # = 30

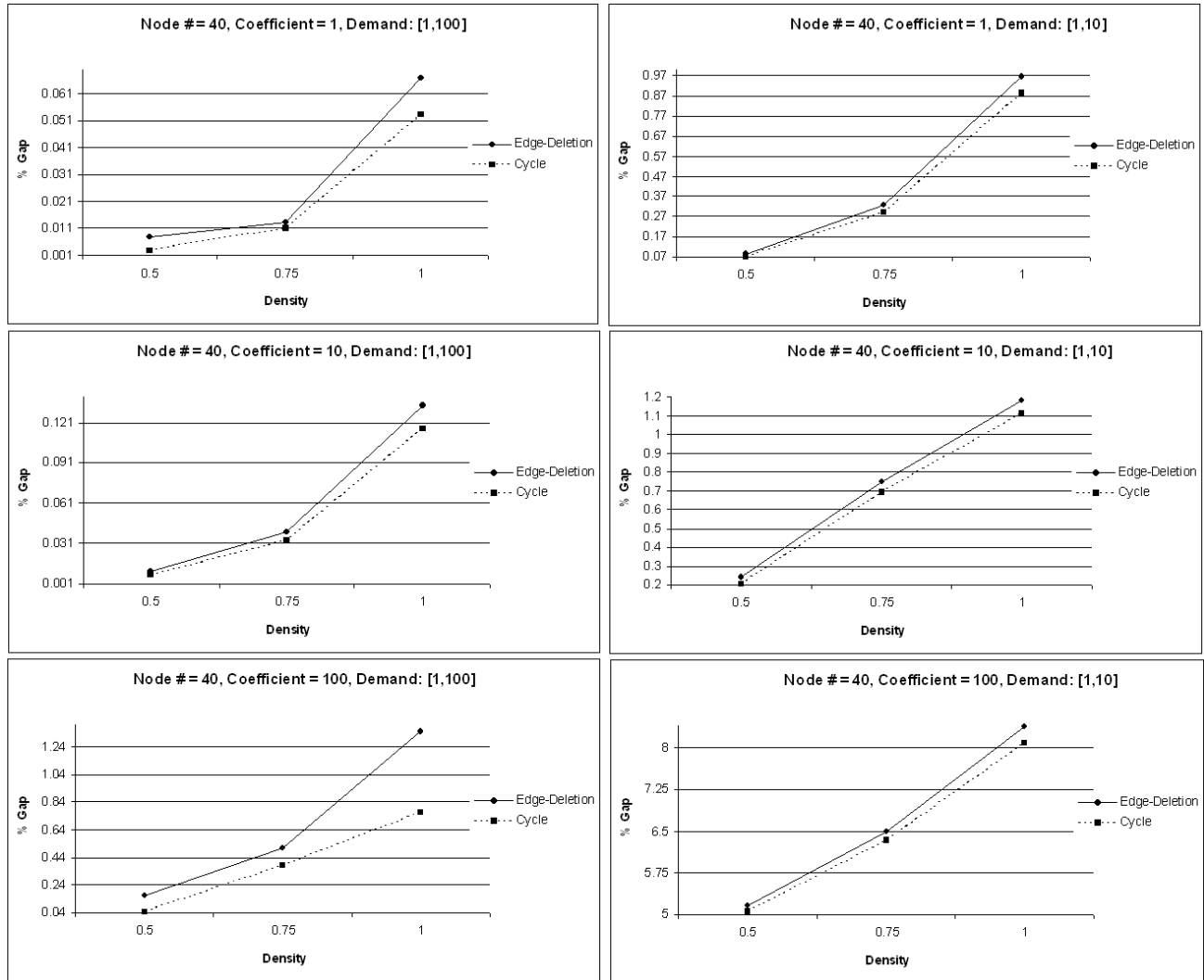


Figure 5.6: Edge Deletion Heuristic and Cycle Algorithm Results for node # = 40

In Figure 5.8 we have unified the graphs for the coefficient value c for node $\# = 30$ and $\# = 40$ to compare the edge deletion heuristic and the cycle algorithm. Similar to the results obtained in Figure 5.7, when $d \in [1, 10]$ the %gap's are very high compared to the instances with $d \in [1, 100]$. But, even when $d \in [1, 10]$, the improvement heuristics make the %gap's considerably smaller compared to the construction heuristic results. The %gap's go from %50's to %10's for the worst cases. The cycle algorithm presents much better %gap values when weighed against the edge deletion heuristic since the cycle algorithm also includes the processes exercised in the edge deletion heuristic.

To sum up, we can conclude by saying that the best results are obtained if the one-step-2 algorithm is employed as the construction heuristic and the cycle algorithm is used as the improvement heuristic.

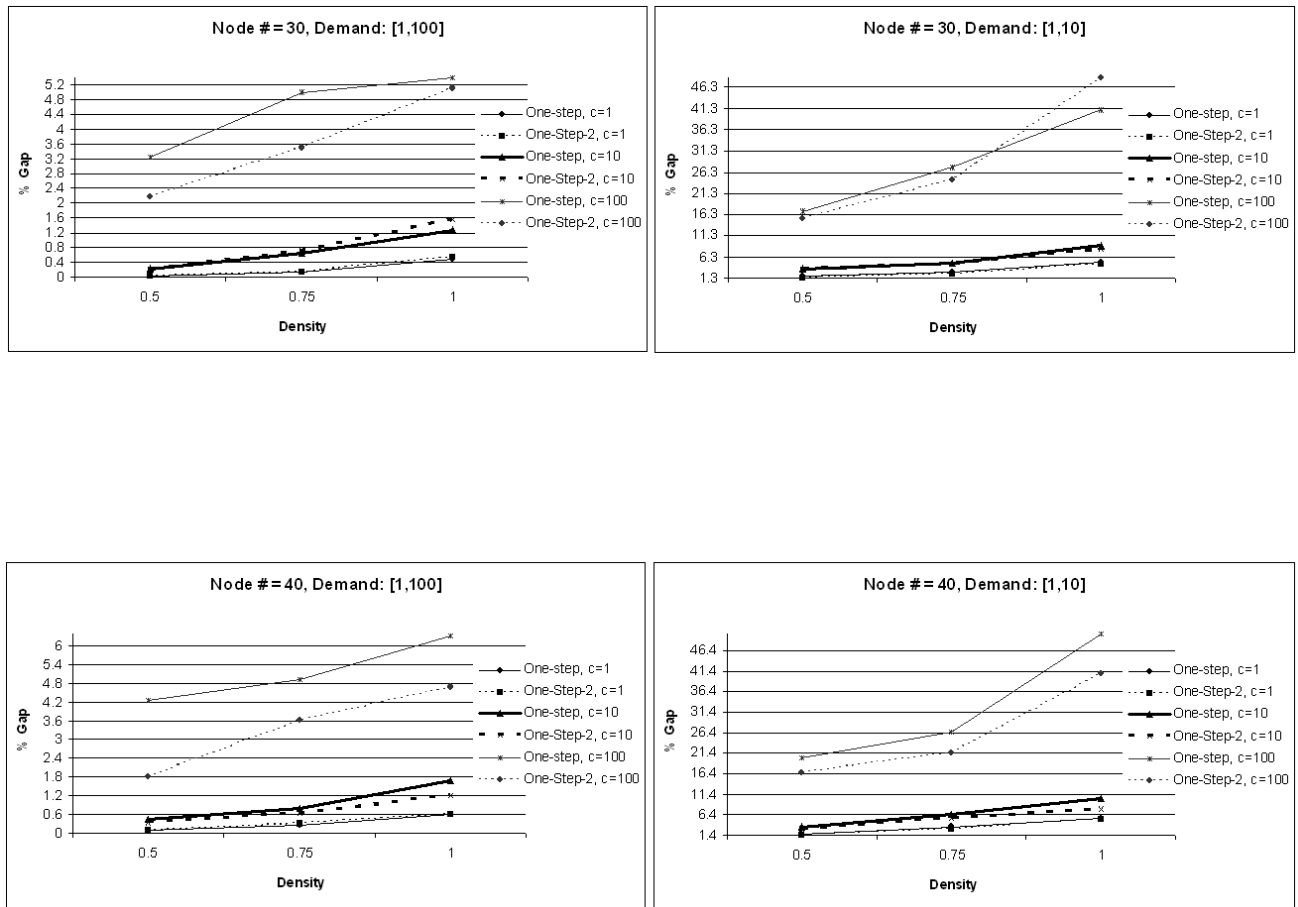


Figure 5.7: One-step and One-step-2 Algorithm: Results for All Test Instances

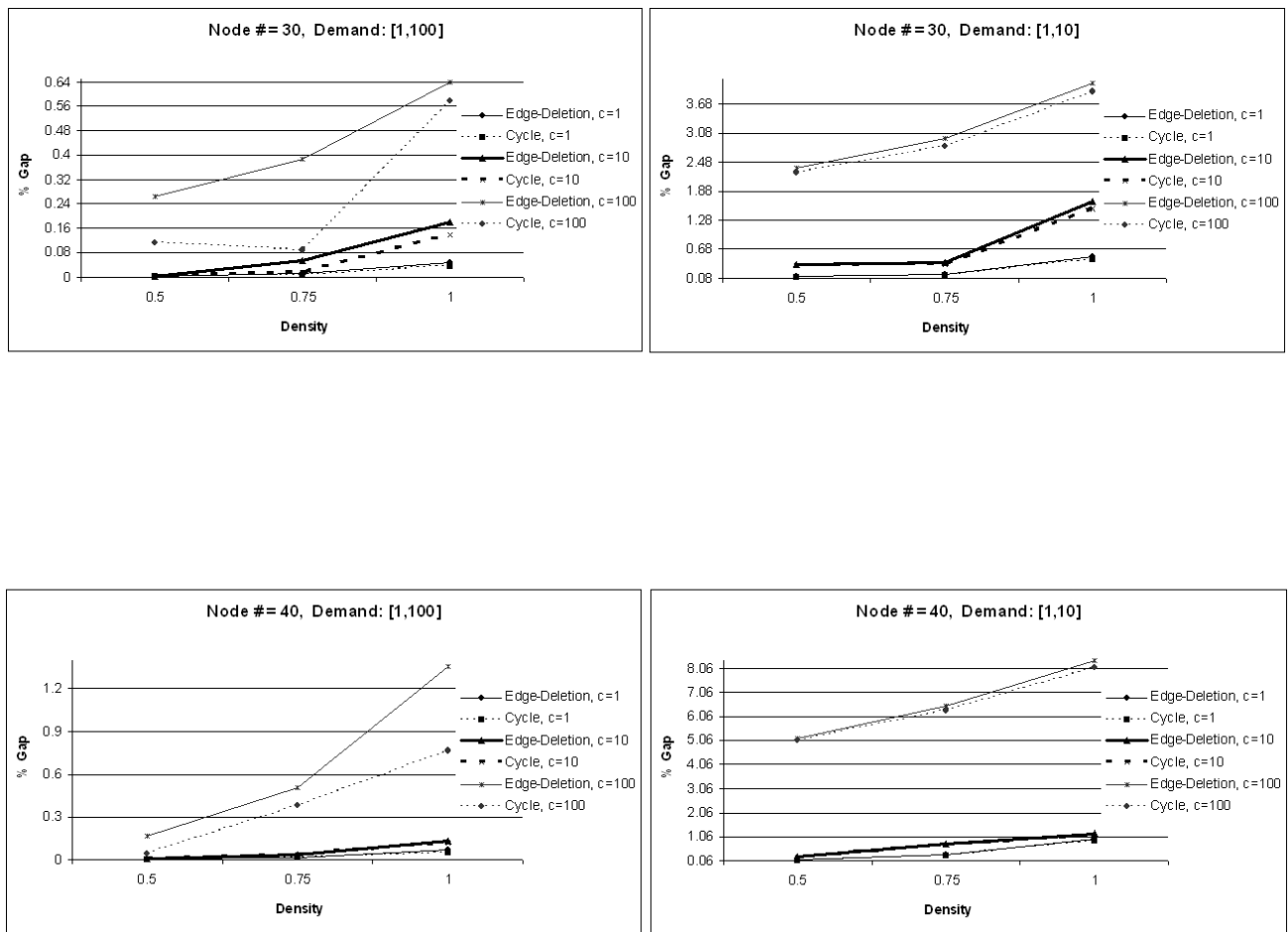


Figure 5.8: Edge Deletion Heuristic and Cycle Algorithm: Results for All Test Instances

Chapter 6

Conclusion

In this thesis, we study algorithms to solve a survivable network design problem arising in telecommunications networks. 2-edge disjoint paths for each possible source and destination pair given a graph G are desired for the survivability requirement. These paths are categorized as primary and secondary paths and have different cost values for the edges. The cost of an edge if used on a primary path is 2 times the cost of an edge used on a secondary path. In addition to routing costs, fixed costs for each edge are also parameters of the problem. Each commodity has demand requests.

An IP model for the proposed problem is developed in Chapter 3. The objective is to minimize the total costs while finding 2-edge disjoint paths for each commodity. Although, problems with differing arc costs for different paths are available in the literature, problems including additional fixed costs are not considered in the literature according to our knowledge.

In Chapter 4, construction heuristics and improvement heuristics that solve the 2-edge disjoint paths problem is presented. Two-step and one-step algorithms are the construction heuristics. The two-step algorithm uses two applications of the Dijkstra's algorithm [1] while one-step algorithm uses two applications of the Suurballe's algorithm [12]. There are four types of improvement heuristics: IP

based heuristic, edge deletion, edge addition and cycle algorithm. The best improvement heuristic in terms of total costs that are in proximity of the optimal value is the cycle-algorithm. This algorithm tries to find new cycles including source and destination pairs by utilizing one-step version 2 algorithm. Improvement edge deletion tries to reduce the number of active edges by inactivating each edge one by one and solving the problem again. Improvement edge addition is the reverse of edge-deletion; since in this method the inactive edges are activated one by one and the problem is solved again. For both improvement methods during the calculations of the new solutions the current solution is updated if the solutions found by the improvement methods are better in terms of proximity to the optimal solution. Furthermore, for improvement edge deletion feasibility conditions are also checked since closing an edge may make the problem infeasible.

Future work for this thesis can include comparing different routing cost structures. Specifically instead of assuming that $1/2ce^1 = ce^2$, new α values that are different than $1/2$ can be tested with the methods explained in Chapter 3 and Chapter 4. Additionally, cases in which the secondary path costs are larger than primary path costs can also be tested. One can also try installing capacities on edges such that more than some demand value cd_{ij} for edge $\{i, j\}$ cannot be sent through that edge. Moreover, new algorithms that are based on a tabu search heuristic can be developed to solve the proposed problem.

Bibliography

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, New Jersey 07458, 1993.
- [2] R. Bhandari. *Survivable Networks: Algorithms for Diverse Routing*. Kluwer Academic Publishers, Boston, 1999.
- [3] R. Bhatia, M. Kodialam, and T. Lakshman. Finding disjoint paths with related path costs. *Journal of Combinatorial Optimization*, 12:83–96, 2006.
- [4] T. Gomes, J. Craveirinha, and L. Jorge. An effective algorithm for obtaining the minimal cost pair of disjoint paths with dual arc costs. *Computers and Operations Research*, 36:1670–1682, 2009.
- [5] T. Gomes, J. Craveirinha, and L. Jorge. An effective algorithm for obtaining the whole set of minimal cost pairs of disjoint paths with dual arc costs. *Journal of Combinatorial Optimization*, 19:394–414, 2010.
- [6] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67:473–496, 2001.
- [7] P. H. Ho, J. Tapolcai, and H. T. Mouftah. On achieving optimal survivable routing for shared protection in survivable next-generation internet. *IEEE Transactions on Reliability*, 53:216–225, 2004.

- [8] A. Itai, Y. Perl, and Y. Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12:277–286, 1982.
- [9] C. L. Li, S. McCormick, and D. Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Appl. Math.*, 26:(1), 1990.
- [10] C. L. Li, S. McCormick, and D. Simchi-Levi. Finding disjoint paths with different path-costs: complexity and algorithms. *Networks*, 22:653–667, 1992.
- [11] D. Ronen and Y. Perl. Heuristics for finding a maximum number of disjoint bounded paths. *Networks*, 14:531–544, 1984.
- [12] J. Suurballe and R. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325–336, 1984.
- [13] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.
- [14] D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He. On finding disjoint paths in single and dual link cost networks. *IEEE INFOCOM*, 2004.

Appendix A

Results of Computational Studies

Tables A.1 to A.6 show the results of one-step and one-step version 2 algorithm with demand $\in \{1, 100\}$ (along with rerouting method) and tables A.7 to A.12 show the results of one-step and one-step version 2 algorithm with demand $\in \{1, 10\}$.

Tables A.13 to A.18 show the results for the edge deletion heuristic and cycle algorithm with demand $\in \{1, 100\}$. Tables A.19 to A.24 show the results for the edge deletion heuristic and cycle algorithm with demand $\in \{1, 10\}$. For both improvement heuristics one-step version 2 algorithm is employed as the construction heuristic. For comparison the optimal solution obtained by the mathematical model described in Chapter 3 is also provided.

Tables A.25 to A.42 show the results of one-step and one-step version 2 algorithm along with reroutings and improvement methods listed in Chapter 4. The tables A.25 to A.27 present instances with 30 nodes along with different coefficient c values and edge densities for demand values ranging from 1 to 100. For each edge density there are three different networks. The instances are solved using one-step algorithm and rerouting after the algorithm ends and also for the edge deletion heuristic. The same tests are also conducted for the 40 node case (next three Tables A.28 to A.30) for one-step algorithm, reroutings and improvement methods.

Using the one-step version 2 algorithm for the settings explained in Chapter 5, Tables A.31 to A.36 present the results. The tables A.37 to A.42 show the results when one-step version 2 algorithm is utilized along with the same settings that differ only in demand values. For the last six tables the demand values range from 1 to 10.

According to the test results for the same settings of the test instances for both one-step and one-step version 2 algorithms (Tables A.25 to A.30 for one-step algorithm and Tables A.31 to A.36 for one-step version 2 algorithm), after the improvements' results, starting with a feasible solution provided by the one-step version 2 algorithm ultimately led to better solutions compared to starting with feasible solutions obtained by the one-step algorithm. After using the one-step version 2 algorithm, the %gap between the optimal solutions and the result of the improvements, especially edge deletion heuristic and cycle algorithm has become considerably small.

However, when the demand values' ranges were reduced to $[1, 10]$ from $[1, 100]$ the %gap even for the one-step version 2 algorithm increased.

Finally, the reader may notice that the results of improvement edge addition are always the same as results of rerouting after an algorithm which implies that no new edge is opened after it has become inactive.

The structure of each table in this section is the same. The first column presents the three different density levels for the edge number calculations. The second column gives the instance number along with the edge number for that instance. The remaining columns present the edges that are activated (edge #), the resource used (running time in seconds) and the %gap calculated according to the optimal solution for the algorithm and/or improvement results. If the optimal solution was not found due to memory or resource restrictions, the %gap is calculated according to the best solution found by the IP model.

Note that instances marked with * do not have optimal solutions due to memory or resource restrictions (RLE stands for Resource Limit Exceeded). For these instances we use the lower bound obtained from the IP model's solution.

Table A.1: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	192	200	187
		0.09	0.01	15.83
		0.04	0.05	—
	2,201	187	190	178
		0.09	0.11	15.11
		0.02	0.04	—
3,204	189	196	187	
	0.08	0.10	14.90	
	0.03	0.03	—	
0.75	1,320	258	279	234
		0.09	0.09	32.10
		0.13	0.03	—
	2,309	262	279	221
		0.09	0.10	27.40
		0.15	0.23	—
3,316	264	285	233	
	0.10	0.10	46.67	
	0.18	0.20	—	
1.0	1,435	334	361	261
		0.08	0.08	99.36
		0.51	0.53	—
	2,435	320	347	234
		0.08	0.08	43.78
		0.47	0.54	—
3,435	343	372	257	
	0.08	0.08	96.48	
	0.50	0.56	—	

Table A.2: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	167	179	155
		0.09	0.10	15.74
		0.22	0.20	—
	2,201	164	174	152
		0.09	0.10	15.10
		0.19	0.19	—
	3,204	169	177	150
		0.08	0.08	14.82
		0.27	0.20	—
0.75	1,320	217	234	177
		0.09	0.09	32.73
		0.53	0.64	—
	2,309	213	236	173
		0.09	0.10	27.72
		0.68	0.76	—
	3,316	223	240	174
		0.09	0.10	27.86
		0.72	0.76	—
1.0	1,435	246	191	181
		0.08	0.09	125.29
		1.39	1.61	—
	2,435	237	283	173
		0.08	0.10	44.78
		1.17	1.47	—
	3,435	251	295	184
		0.08	0.09	127.84
		1.26	1.58	—

Table A.3: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	87	130	91
		0.08	0.10	40.06
		2.65	2.55	—
	2,201	94	134	101
		0.08	0.10	16.16
		2.81	2.03	—
	3,204	102	129	97
		0.08	0.10	16.02
		4.32	1.94	—
0.75	1,320	105	147	103
		0.08	0.09	34.36
		5.37	3.84	—
	2,309	99	147	105
		0.09	0.09	30.31
		5.67	3.12	—
	3,316	108	155	105
		0.08	0.08	84.07
		3.94	3.54	—
1.0	1,435	108	157	103
		0.08	0.08	352.04
		5.23	5.04	—
	2,435	98	172	102
		0.08	0.08	251.83
		5.54	6.31	—
	3,435	105	158	105
		0.08	0.08	49.08
		5.40	4.03	—

Table A.4: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	350	366	316
		0.30	0.31	125.67
		0.08	0.09	—
	2,383	331	352	296
		0.29	0.28	120.60
		0.10	0.12	—
3,373	341	349	313	
	0.29	0.32	166.32	
	0.08	0.08	—	
0.75	1,580	467	491	362
		0.30	0.30	198.89
		0.31	0.32	—
	2,580	432	462	352
		0.30	0.28	197.17
		0.31	0.30	—
	3,576	449	483	369
		0.30	0.30	246.74
		0.20	0.24	—
*1.0	1,780	565	613	Out of Memory
		0.28	0.28	325.77
		0.52	0.64	—
	2,780	531	604	Out of Memory
		0.28	0.26	298.90
		0.70	0.67	—
	3,780	573	618	Out of Memory
		0.28	0.26	301.46
		0.58	0.54	—

Table A.5: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	300	305	246
		0.29	0.28	134.51
		0.54	0.33	—
	2,383	276	301	243
		0.28	0.29	322.92
		0.37	0.35	—
3,373	279	301	238	
	0.34	0.34	117.59	
	0.39	0.32	—	
0.75	1,580	341	375	272
		0.30	0.29	213.79
		0.87	0.72	—
	2,580	336	365	263
		0.29	0.30	672.32
		0.86	0.69	—
3,576	326	359	278	
	0.30	0.28	704.02	
	0.70	0.51	—	
*1.0	1,780	365	442	Out of Memory
		0.28	0.28	307.21
		1.62	1.27	—
	2,780	377	439	Out of Memory
		0.28	0.28	313.94
		1.65	1.17	—
3,780	377	436	Out of Memory	
	0.28	0.28	309.79	
	1.79	1.15	—	

Table A.6: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	139	205	151
		0.27	0.28	427.90
		3.86	1.90	—
	2,383	143	208	149
		0.28	0.28	126.49
		4.56	1.92	—
3,373	126	194	142	
	0.27	0.29	361.56	
	4.30	1.57	—	
0.75	1,580	146	242	154
		0.27	0.30	1287.57
		4.19	4.01	—
	2,580	140	227	155
		0.28	0.27	1307.09
		5.57	2.64	—
3,576	136	251	156	
	0.28	0.28	1103.88	
	4.99	4.20	—	
*1.0	1,780	133	258	Out of Memory
		0.25	0.29	455.46
		6.09	5.38	—
	2,780	142	241	Out of Memory
		0.25	0.26	576.27
		5.93	3.97	—
3,780	134	255	Out of Memory	
	0.27	0.29	717.44	
	6.99	4.76	—	

Table A.7: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 1, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	144	172	121
		0.08	0.08	16.43
		2.05	1.53	—
	2,201	131	165	120
		0.08	0.08	46.98
		1.35	1.22	—
3,204	145	170	122	
	0.08	0.08	15.34	
	1.77	1.35	—	
0.75	1,320	179	211	136
		0.10	0.11	33.91
		2.57	2.35	—
	2,309	152	198	134
		0.08	0.08	72.50
		2.72	2.42	—
3,316	172	217	137	
	0.07	0.09	87.23	
	2.69	2.41	—	
1.0	1,435	175	254	136
		0.08	0.09	172.89
		4.71	4.27	—
	2,435	166	247	128
		0.08	0.08	53.27
		5.12	4.83	—
3,435	192	266	144	
	0.08	0.08	79.90	
	4.83	4.54	—	

Table A.8: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 10, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	89	135	87
		0.08	0.09	48.75
		3.92	3.40	—
	2,201	90	132	93
		0.07	0.08	16.44
		3.51	3.09	—
3,204	94	131	88	
	0.05	0.08	15.82	
	3.24	3.01	—	
0.75	1,320	94	154	97
		0.09	0.09	115.05
		5.02	4.84	—
	2,309	95	141	98
		0.07	0.08	188.58
		3.91	3.60	—
3,316	105	158	98	
	0.08	0.09	31.47	
	4.98	4.83	—	
1.0	1,435	97	165	95
		0.07	0.08	56.68
		7.35	6.99	—
	2,435	91	186	93
		0.07	0.08	3604.11
		10.88	10.39	—
3,435	103	177	102	
	0.07	0.08	74.14	
	8.47	7.32	—	

Table A.9: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 30, Coefficient = 100, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	53	90	51
		0.07	0.08	226.68
		18.12	16.91	—
	2,201	51	95	55
		0.08	0.10	3601.96
		19.21	17.72	—
3,204	54	89	58	
	0.07	0.08	640.36	
	13.91	11.11	—	
0.75	1,320	55	107	57
		0.08	0.08	3603.20
		28.01	24.41	—
	2,309	55	112	55
		0.09	0.10	3616.24
		35.11	32.09	—
3,316	54	103	57	
	0.12	0.08	3602.83	
	19.37	17.13	—	
1.0	1,435	56	143	57
		0.12	0.08	3603.82
		41.90	62.88	—
	2,435	66	137	57
		0.09	0.08	3603.76
		50.19	54.21	—
3,435	60	119	58	
	0.08	0.08	3603.20	
	30.59	28.58	—	

Table A.10: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 1, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	237	275	184
		0.32	0.28	127.19
		1.60	1.43	—
	2,383	207	266	185
		0.32	0.28	126.10
		1.67	1.48	—
3,373	231	276	194	
	0.31	0.28	122.41	
	1.63	1.31	—	
0.75	1,580	251	345	193
		0.32	0.29	872.01
		3.61	3.22	—
	2,580	238	342	194
		0.30	0.28	1297.85
		3.84	3.46	—
3,576	279	346	206	
	0.30	0.29	854.12	
	2.56	2.34	—	
*1.0	1,780	278	419	Out of Memory
		0.28	0.29	359.60
		5.38	5.04	—
	2,780	259	402	Out of Memory
		0.28	0.28	102.05
		5.06	4.69	—
3,780	306	441	Out of Memory	
	0.29	0.28	721.88	
	5.92	5.54	—	

Table A.11: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 10, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	130	207	144
		0.32	0.27	711.18
		3.57	3.01	—
	2,383	126	206	132
		0.32	0.27	147.80
		3.96	3.16	—
3,373	149	202	135	
	0.31	0.27	448.13	
	3.11	2.65	—	
0.75	1,580	130	241	145
		0.32	0.28	1066.10
		6.09	5.19	—
	2,580	125	242	134
		0.30	0.30	3609.27
		7.24	5.99	—
3,576	143	250	147	
	0.32	0.28	3188.68	
	6.13	5.37	—	
*1.0	1,780	133	290	Out of Memory
		0.28	0.26	1088.13
		9.75	8.27	—
	2,780	135	262	Out of Memory
		0.28	0.25	2693.73
		9.37	6.19	—
3,780	143	284	Out of Memory	
	0.32	0.26	1778.69	
	12.15	8.45	—	

Table A.12: Computational Results of the One-Step and One-Step-2 Algorithm along with Rerouting, Node # = 40, Coefficient = 100, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	One-Step & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	One-Step-2 & Reroute <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	86 0.32 18.32	143 0.27 15.90	RLE 3607.56 —
	2,383	75 0.32 23.27	147 0.26 18.01	84 3606.05 —
	3,373	82 0.30 19.16	141 0.26 15.85	81 3606.06 —
0.75	1,580	90 0.30 26.68	189 0.27 19.91	RLE 3607.26 —
	2,580	88 0.30 26.24	171 0.28 22.14	RLE 3607.94 —
	3,576	84 0.36 27.06	194 0.27 22.63	RLE 3607.18 —
*1.0	1,780	100 0.26 49.85	212 0.26 40.07	RLE 3609.82 —
	2,780	90 0.28 45.59	178 0.25 30.88	RLE 3609.79 —
	3,780	89 0.32 55.96	212 0.26 51.71	RLE 3609.75 —

Table A.13: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	186	186	187
		10.02	10.10	15.83
		0.01	0.01	—
	2,201	178	178	178
		9.40	9.48	15.11
		0	0	—
	3,204	187	187	187
		9.57	9.64	14.90
		0	0	—
0.75	1,320	229	233	234
		14.04	14.25	32.10
		0.01	0.004	—
	2,309	219	220	221
		14.40	14.17	27.40
		0.01	0.003	—
	3,316	227	230	233
		14.09	14.09	46.67
		0.02	0.01	—
1.0	1,435	244	249	261
		17.31	17.58	99.36
		0.07	0.06	—
	2,435	227	229	234
		16.92	16.92	43.78
		0.05	0.04	—
	3,435	247	251	257
		17.68	18.05	96.48
		0.02	0.01	—

Table A.14: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	154	155	155
		8.77	8.85	15.74
		0.004	0.002	—
	2,201	152	152	152
		8.40	8.51	15.10
		0	0	—
	3,204	149	149	150
		8.42	8.52	14.82
		0.01	0.01	—
0.75	1,320	167	171	177
		11.54	11.70	32.73
		0.06	0.03	—
	2,309	169	172	173
		11.63	11.81	27.72
		0.04	0.01	—
	3,316	171	173	174
		11.78	11.98	27.86
		0.06	0.01	—
1.0	1,435	172	174	181
		14.15	14.35	125.29
		0.22	0.18	—
	2,435	161	167	173
		13.94	14.16	44.78
		0.17	0.12	—
	3,435	173	178	184
		14.16	14.36	127.84
		0.15	0.12	—

Table A.15: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	87	90	91
		5.96	6.08	40.06
		0.30	0.13	—
	2,201	99	99	101
		6.10	6.25	16.16
		0.24	0.10	—
	3,204	96	96	97
		5.78	5.89	16.02
		0.26	0.11	—
0.75	1,320	99	101	103
		6.82	6.99	34.36
		0.23	0.10	—
	2,309	98	100	105
		6.78	6.94	30.31
		0.21	0.09	—
	3,316	112	106	105
		7.10	7.30	84.07
		0.72	0.08	—
1.0	1,435	100	101	103
		7.35	7.41	352.04
		0.48	0.40	—
	2,435	96	98	102
		8.39	8.15	251.83
		0.91	0.88	—
	3,435	100	102	105
		7.53	7.32	49.08
		0.53	0.46	—

Table A.16: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	313	315	316
		58.60	59.57	125.67
		0.003	0.001	—
	2,383	295	296	296
		55.16	56.25	120.60
		0.01	0.003	—
	3,373	309	311	313
		55.87	57.04	166.32
		0.01	0.004	—
0.75	1,580	355	357	362
		79.38	80.02	198.89
		0.02	0.02	—
	2,580	353	350	352
		81.11	74.34	197.17
		0.01	0.003	—
	3,576	367	368	369
		78.45	79.94	246.74
		0.01	0.01	—
*1.0	1,780	373	376	Out of Memory
		96.67	96.67	325.77
		0.08	0.07	—
	2,780	370	372	Out of Memory
		93.67	98.82	298.90
		0.07	0.06	—
	3,780	385	386	Out of Memory
		96.87	99.03	301.46
		0.05	0.03	—

Table A.17: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	245	245	246
		47.12	48.05	134.51
		0.001	0.001	—
	2,383	241	242	243
		45.83	46.64	322.92
		0.02	0.02	—
	3,373	234	236	238
		46.68	47.86	117.59
		0.01	0.002	—
0.75	1,580	262	265	272
		58.71	60.29	213.79
		0.06	0.05	—
	2,580	255	257	263
		55.83	57.18	672.32
		0.04	0.03	—
	3,576	269	274	278
		56.21	57.42	704.02
		0.02	0.02	—
*1.0	1,780	257	259	Out of Memory
		69.07	70.92	307.21
		0.11	0.10	—
	2,780	266	268	Out of Memory
		67.74	68.67	313.94
		0.12	0.11	—
	3,780	267	269	Out of Memory
		68.26	69.51	309.79
		0.17	0.14	—

Table A.18: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	149	150	151
		29.47	30.52	427.90
		0.13	0.04	—
	2,383	141	142	149
		29.48	30.23	126.49
		0.22	0.06	—
	3,373	139	141	142
		27.95	28.91	361.56
		0.15	0.04	—
0.75	1,580	146	148	154
		35.39	36.54	1287.57
		0.35	0.31	—
	2,580	152	153	155
		32.73	33.41	1307.09
		0.34	0.23	—
	3,576	156	155	156
		36.87	36.87	1103.88
		0.83	0.60	—
*1.0	1,780	177	176	Out of Memory
		38.27	38.75	455.46
		1.74	1.05	—
	2,780	151	153	Out of Memory
		34.97	35.80	576.27
		0.99	0.69	—
	3,780	168	164	Out of Memory
		37.75	38.21	717.44
		1.33	0.56	—

Table A.19: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> ---
0.5	1,212	116	119	121
		7.29	8.33	16.43
		0.17	0.15	—
	2,201	115	116	120
		7.12	7.79	46.98
		0.13	0.12	—
	3,204	115	117	122
		6.91	7.88	15.34
		0.06	0.06	—
0.75	1,320	128	130	136
		10.04	10.35	33.91
		0.10	0.08	—
	2,309	123	125	134
		8.72	10.03	72.50
		0.24	0.21	—
	3,316	130	131	137
		9.83	10.38	87.23
		0.13	0.12	—
1.0	1,435	124	125	136
		11.48	12.33	172.89
		0.57	0.51	—
	2,435	118	123	128
		10.81	12.11	53.27
		0.51	0.44	—
	3,435	131	131	144
		12.10	12.80	79.90
		0.48	0.46	—

Table A.20: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> ---
0.5	1,212	84	86	87
		5.14	6.26	48.75
		0.45	0.39	—
	2,201	88	89	93
		5.92	6.07	16.44
		0.51	0.47	—
	3,204	85	88	88
		5.12	5.87	15.82
		0.14	0.13	—
0.75	1,320	92	95	97
		6.28	7.16	115.05
		0.44	0.40	—
	2,309	90	90	98
		6.07	6.54	188.58
		0.64	0.59	—
	3,316	97	99	98
		6.51	7.22	31.47
		0.12	0.08	—
1.0	1,435	93	100	95
		5.93	7.73	56.68
		1.19	1.02	—
	2,435	90	90	93
		7.14	8.72	3604.11
		1.97	1.82	—
	3,435	95	96	102
		7.26	8.13	74.14
		1.85	1.72	—

Table A.21: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,212	52	50	51
		3.18	3.94	226.68
		2.33	2.28	—
	2,201	50	50	55
		3.58	4.01	3601.96
		2.54	2.41	—
	3,204	54	55	58
		3.29	3.88	640.36
		2.17	2.11	—
0.75	1,320	52	55	57
		4.28	4.87	3603.20
		2.83	2.66	—
	2,309	49	51	55
		4.82	5.12	3616.24
		3.18	3.07	—
	3,316	55	58	57
		4.80	5.24	3602.83
		2.91	2.68	—
1.0	1,435	52	58	57
		5.04	6.57	3603.82
		4.48	4.32	—
	2,435	53	61	57
		4.62	6.19	3603.76
		4.31	4.06	—
	3,435	55	57	58
		4.47	5.24	3603.20
		3.58	3.42	—

Table A.22: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	183	184	184
		40.51	41.14	127.19
		0.10	0.09	—
	2,383	179	183	185
		39.24	39.31	126.10
		0.08	0.06	—
	3,373	185	187	194
		40.12	41.88	122.41
		0.07	0.06	—
0.75	1,580	179	182	193
		51.86	53.07	872.01
		0.26	0.23	—
	2,580	170	173	194
		49.93	51.27	1297.85
		0.45	0.41	—
	3,576	196	199	206
		52.34	53.41	854.12
		0.28	0.23	—
*1.0	1,780	186	188	Out of Memory
		64.10	64.55	359.60
		0.88	0.82	—
	2,780	180	182	Out of Memory
		59.22	60.75	102.05
		0.97	0.90	—
	3,780	185	185	Out of Memory
		67.29	68.38	721.88
		1.05	0.94	—

Table A.23: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	133	137	144
		28.21	29.76	711.18
		0.35	0.31	—
	2,383	127	130	132
		27.61	29.05	147.80
		0.19	0.16	—
	3,373	131	135	135
		27.85	29.29	448.13
		0.19	0.15	—
0.75	1,580	139	144	145
		33.52	35.34	1066.10
		0.65	0.59	—
	2,580	130	134	134
		33.09	34.81	3609.27
		0.83	0.76	—
	3,576	137	140	147
		35.26	36.79	3188.68
		0.76	0.72	—
*1.0	1,780	137	138	Out of Memory
		41.53	42.90	1088.13
		1.18	1.14	—
	2,780	129	134	Out of Memory
		35.72	37.95	2693.73
		1.22	1.14	—
	3,780	134	136	Out of Memory
		40.16	41.81	1778.69
		1.14	1.06	—

Table A.24: Computational Results of the Edge-Deletion Heuristic and Cycle Algorithm Applied After One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 10\}$

Density	Instances, <i>edge#</i>	Edge-Deletion <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Cycle <i>edge#</i> <i>time(sec.)</i> <i>%gap</i>	Optimal Sol. <i>edge#</i> <i>time(sec.)</i> — — —
0.5	1,390	78	80	RLE
		17.32	18.44	3607.56
		5.13	5.02	—
	2,383	103	105	84
		16.12	17.25	3606.05
		6.22	6.14	—
	3,373	75	81	81
		17.58	19.10	3606.06
		4.12	4.01	—
0.75	1,580	74	99	RLE
		21.84	24.52	3607.26
		5.46	5.24	—
	2,580	79	79	RLE
		24.06	24.15	3607.94
		6.89	6.76	—
	3,576	85	96	RLE
		22.16	25.83	3607.18
		7.14	7.02	—
*1.0	1,780	83	90	RLE
		27.38	30.16	3609.82
		8.01	7.54	—
	2,780	83	86	RLE
		24.50	25.65	3609.79
		7.65	7.42	—
	3,780	85	91	RLE
		28.87	30.47	3609.75
		9.48	9.32	—

Table A.25: Computational Results of the One-Step Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step edge# time(sec.) %gap	IP Based Heuristic edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge-Del. edge# time(sec.) %gap	Cycle edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) -- --	
0.5	1,212	192	192	192	184	184	187	
		0.05	4.99	0.09	9.24	9.71	15.83	
		0.13	0.04	0.04	0.02	0.02	—	
	2,201	187	187	187	178	178	178	178
		0.06	4.63	0.09	8.22	9.49	15.11	
		0.09	0.02	0.02	0	0	—	
	3,204	189	189	189	185	185	187	187
		0.04	4.69	0.08	8.45	9.33	14.90	
		0.10	0.03	0.03	0.01	0.01	—	
0.75	1,320	258	258	258	225	226	234	
		0.06	6.88	0.09	13.10	13.30	32.10	
		0.23	0.13	0.13	0.03	0.01	—	
	2,309	262	262	262	220	220	221	221
		0.06	6.87	0.09	14.22	14.35	27.40	
		0.33	0.15	0.15	0.02	0.01	—	
	3,316	264	263	264	226	225	233	233
		0.05	6.88	0.10	13.25	13.46	46.67	
		0.36	0.17	0.18	0.04	0.01	—	
1.0	1,435	334	334	334	252	253	261	
		0.04	9.22	0.08	16.14	17.26	99.36	
		0.71	0.51	0.51	0.13	0.07	—	
	2,435	320	319	320	227	229	234	234
		0.04	12.27	0.08	15.02	16.67	43.78	
		0.69	0.46	0.47	0.08	0.05	—	
	3,435	343	341	343	246	247	257	257
		0.05	9.18	0.08	15.81	16.42	96.48	
		0.72	0.48	0.50	0.05	0.03	—	

Table A.26: Computational Results of the One-Step Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step edge# time(sec.) %gap	IP Based Heuristic edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge-Del. edge# time(sec.) %gap	Cycle edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) -- --
0.5	1,212	167	167	167	149	151	155
		0.04	4.52	0.09	8.12	8.83	15.74
		0.52	0.21	0.22	0.05	0.01	—
	2,201	164	164	164	148	149	152
		0.04	4.44	0.09	7.89	8.14	15.10
		0.47	0.19	0.19	0.04	0.02	—
	3,204	169	169	169	147	146	150
		0.04	4.41	0.08	6.98	7.56	14.82
		0.49	0.26	0.27	0.05	0.01	—
0.75	1,320	217	217	217	169	171	177
		0.04	6.41	0.09	10.03	11.32	32.73
		0.84	0.53	0.53	0.06	0.03	—
	2,309	213	213	213	167	168	173
		0.04	6.28	0.09	10.55	10.94	27.72
		1.02	0.68	0.68	0.12	0.05	—
	3,316	223	223	223	172	172	174
		0.05	6.34	0.09	11.24	11.32	27.86
		1.10	0.72	0.72	0.08	0.02	—
1.0	1,435	246	246	246	162	165	181
		0.04	8.19	0.08	13.67	13.88	125.29
		1.75	1.39	1.39	0.31	0.27	—
	2,435	237	237	237	160	160	173
		0.06	7.96	0.08	12.30	13.13	44.78
		1.67	1.17	1.17	0.27	0.20	—
	3,435	251	251	251	172	178	184
		0.04	8.15	0.08	12.63	13.52	127.84
		1.82	1.26	1.26	0.30	0.21	—

Table A.27: Computational Results of the One-Step Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, $edge\#$	One-Step $edge\#$ $time(sec.)$ $\%gap$	IP Based Heuristic $edge\#$ $time(sec.)$ $\%gap$	Reroute $edge\#$ $time(sec.)$ $\%gap$	Edge-Del. $edge\#$ $time(sec.)$ $\%gap$	Cycle $edge\#$ $time(sec.)$ $\%gap$	Optimal Sol. $edge\#$ $time(sec.)$ $-- --$	
0.5	1,212	87	87	87	80	82	91	
		0.05	3.76	0.08	3.29	4.00	40.06	
		6.07	2.64	2.65	2.04	1.73	—	
	2,201	94	94	94	84	84	84	101
		0.04	3.67	0.08	2.57	4.03	4.03	16.16
		5.21	2.80	2.81	1.56	1.34	—	—
	3,204	102	102	102	86	86	85	97
		0.07	3.81	0.08	3.14	4.68	4.68	16.02
		8.20	4.32	4.32	2.49	2.15	—	—
0.75	1,320	105	105	105	86	90	103	
		0.05	5.15	0.08	4.07	4.96	4.96	34.36
		8.16	5.36	5.37	3.38	3.12	—	—
	2,309	99	99	99	83	86	86	105
		0.05	4.96	0.09	4.21	4.60	4.60	30.31
		9.04	5.67	5.67	4.26	4.13	—	—
	3,316	108	108	108	88	90	90	105
		0.05	5.07	0.08	3.71	4.95	4.95	84.07
		7.07	3.94	3.94	2.31	2.06	—	—
1.0	1,435	108	108	108	85	91	103	
		0.04	6.47	0.08	4.91	5.04	5.04	352.04
		10.50	5.14	5.23	2.98	2.63	—	—
	2,435	98	98	98	83	85	85	102
		0.04	6.36	0.08	4.05	4.54	4.54	251.83
		8.95	5.54	5.54	3.66	3.34	—	—
	3,435	105	105	105	90	89	89	105
		0.06	6.32	0.08	4.22	4.80	4.80	49.08
		9.81	5.40	5.40	3.34	3.22	—	—

Table A.28: Computational Results of the One-Step Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step edge# time(sec.) %gap	IP Based Heuristic edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge-Del. edge# time(sec.) %gap	Cycle edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	350	350	350	312	313	316
		0.17	18.62	0.30	56.22	57.56	125.67
		0.22	0.08	0.08	0.01	0.004	—
	2,383	331	331	331	291	293	296
		0.15	16.32	0.29	52.22	53.54	120.60
		0.23	0.10	0.10	0.02	0.01	—
3,373	341	341	341	309	309	313	
	0.14	16.34	0.29	55.35	56.12	166.32	
	0.14	0.07	0.08	0.01	0.01	—	
0.75	1,580	467	466	467	356	354	362
		0.18	25.45	0.30	75.13	77.79	198.89
		0.56	0.31	0.31	0.05	0.03	—
	2,580	432	430	432	342	344	352
		0.15	32.82	0.30	71.23	74.44	197.17
		0.53	0.30	0.31	0.05	0.02	—
3,576	449	445	449	358	362	369	
	0.17	23.54	0.30	72.88	74.80	246.74	
	0.38	0.18	0.20	0.02	0.01	—	
*1.0	1,780	565	562	565	372	372	Out of Memory
		0.15	40.43	0.28	91.02	92.86	325.77
		0.78	0.51	0.52	0.08	0.08	—
	2,780	531	524	531	369	372	Out of Memory
		0.14	35.32	0.28	85.02	86.57	298.90
		0.81	0.67	0.70	0.09	0.08	—
3,780	573	567	573	381	382	Out of Memory	
	0.14	30.95	0.28	92.27	93.43	301.46	
	0.68	0.56	0.58	0.08	0.07	—	

Table A.29: Computational Results of the One-Step Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step edge# time(sec.) %gap	IP Based Heuristic edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge-Del. edge# time(sec.) %gap	Cycle edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	300	300	300	240	242	246
		0.17	16.92	0.29	46.84	48.21	134.51
		0.91	0.54	0.54	0.11	0.06	—
	2,383	276	275	276	233	235	243
		0.15	14.98	0.28	41.18	43.36	322.92
		0.75	0.35	0.37	0.07	0.04	—
	3,373	279	278	279	230	233	238
		0.15	15.01	0.34	43.36	45.15	117.59
		0.70	0.38	0.39	0.15	0.08	—
0.75	1,580	341	341	341	252	256	272
		0.17	21.16	0.30	52.33	55.44	213.79
		1.41	0.87	0.87	0.19	0.14	—
	2,580	336	336	336	248	248	263
		0.15	28.10	0.29	51.22	53.58	672.32
		1.36	0.86	0.86	0.22	0.16	—
	3,576	326	326	326	253	254	278
		0.15	20.54	0.30	52.78	54.31	704.02
		1.11	0.69	0.70	0.19	0.11	—
*1.0	1,780	365	364	365	257	261	Out of Memory
		0.14	25.34	0.28	58.44	59.41	307.21
		2.43	1.60	1.62	0.28	0.21	—
	2,780	377	377	377	265	269	Out of Memory
		0.14	34.96	0.28	58.74	60.36	313.94
		2.41	1.62	1.65	0.32	0.27	—
	3,780	377	374	377	252	256	Out of Memory
		0.15	25.96	0.28	59.15	61.03	309.79
		2.52	1.74	1.79	0.26	0.19	—

Table A.30: Computational Results of the One-Step Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step edge# time(sec.) %gap	IP Based Heuristic edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge-Del. edge# time(sec.) %gap	Cycle edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	139	139	139	122	120	151
		0.16	12.35	0.27	19.38	20.22	427.90
	2,383	7.28	3.84	3.86	2.82	2.27	—
		143	143	143	120	122	149
	3,373	0.16	12.26	0.28	20.03	21.01	126.49
		8.27	4.48	4.56	3.05	2.74	—
0.75	1,580	126	126	126	110	110	142
		0.15	11.88	0.27	18.03	19.63	361.56
	2,580	7.22	4.26	4.30	3.36	3.17	—
		146	146	146	129	129	154
	3,576	0.16	16.37	0.27	20.16	21.75	1287.57
		7.68	4.17	4.19	2.91	2.40	—
*1.0	1,780	140	140	140	122	126	155
		0.15	16.46	0.28	19.18	20.50	1307.09
	2,780	9.95	5.56	5.57	4.40	4.27	—
		136	136	136	117	121	156
	3,780	0.15	18.55	0.28	18.28	19.09	1103.88
		7.91	4.99	4.99	3.25	3.12	—
*1.0	1,780	133	133	133	122	121	Out of Memory
		0.14	19.98	0.25	17.86	19.44	455.46
	2,780	9.97	6.04	6.09	2.12	1.94	—
		142	142	142	118	120	Out of Memory
	3,780	0.14	20.15	0.25	18.58	20.84	576.27
		9.94	5.89	5.93	2.25	2.03	—
3,780	134	134	134	117	121	Out of Memory	
	0.14	20.16	0.27	17.86	19.66	717.44	
		10.01	6.94	6.99	2.77	2.32	—

Table A.31: Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,212	200	200	186	200	187
		0.06	0.01	10.02	1.17	15.83
		0.06	0.05	0.01	0.05	—
	2,201	190	190	178	190	178
		0.06	0.11	9.40	1.16	15.11
		0.06	0.04	0	0.04	—
	3,204	196	196	187	196	187
		0.05	0.10	9.57	0.79	14.90
		0.04	0.03	0	0.03	—
0.75	1,320	279	279	229	279	234
		0.04	0.09	14.04	3.74	32.10
		0.20	0.03	0.01	0.03	—
	2,309	279	279	219	279	221
		0.05	0.10	14.40	2.98	27.40
		0.28	0.23	0.01	0.23	—
	3,316	285	285	227	285	233
		0.05	0.10	14.09	3.09	46.67
		0.23	0.20	0.02	0.20	—
1.0	1,435	361	361	244	361	261
		0.04	0.08	17.31	6.33	99.36
		0.60	0.53	0.07	0.53	—
	2,435	347	347	227	347	234
		0.04	0.08	16.92	7.64	43.78
		0.62	0.54	0.05	0.54	—
	3,435	372	372	247	372	257
		0.03	0.08	17.68	5.45	96.48
		0.63	0.56	0.02	0.56	—

Table A.32: Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,212	179	179	154	179	155
		0.06	0.10	8.77	3.01	15.74
		0.30	0.20	0.004	0.20	—
	2,201	174	174	152	174	152
		0.05	0.10	8.40	2.46	15.10
		0.26	0.19	0	0.19	—
	3,204	177	177	149	177	150
		0.04	0.08	8.42	2.45	14.82
		0.28	0.20	0.01	0.20	—
0.75	1,320	234	234	167	234	177
		0.04	0.09	11.54	8.35	32.73
		0.82	0.64	0.06	0.64	—
	2,309	236	236	169	236	173
		0.05	0.10	11.63	6.58	27.72
		0.89	0.76	0.04	0.76	—
3,316	240	240	171	240	174	
	0.05	0.10	11.78	6.88	27.86	
	0.92	0.76	0.06	0.76	—	
1.0	1,435	291	291	172	291	181
		0.04	0.09	14.15	12.80	125.29
		2.12	1.61	0.22	1.61	—
	2,435	283	283	161	283	173
		0.05	0.10	13.94	13.05	44.78
		1.63	1.47	0.17	1.47	—
	3,435	295	295	173	295	184
		0.05	0.09	14.16	11.42	127.84
		1.80	1.58	0.15	1.58	—

Table A.33: Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, $edge\#$	One-Step-2 $edge\#$ $time(sec.)$ $\%gap$	Reroute $edge\#$ $time(sec.)$ $\%gap$	Edge Del. $edge\#$ $time(sec.)$ $\%gap$	Edge Add. $edge\#$ $time(sec.)$ $\%gap$	Optimal Sol. $edge\#$ $time(sec.)$ ---
0.5	1,212	130	130	87	130	91
		0.05	0.10	5.96	7.82	40.06
		3.87	2.55	0.30	2.55	—
	2,201	134	134	99	134	101
		0.06	0.10	6.10	5.85	16.16
		2.72	2.03	0.24	2.03	—
3,204	129	129	96	129	97	
	0.05	0.10	5.78	6.47	16.02	
	2.73	1.94	0.26	1.94	—	
0.75	1,320	147	147	99	147	103
		0.04	0.09	6.82	16.41	34.36
		6.92	3.84	0.23	3.84	—
	2,309	147	147	98	147	105
		0.05	0.09	6.78	14.09	30.31
		4.52	3.12	0.21	3.12	—
3,316	155	155	112	155	105	
	0.04	0.08	7.10	15.45	84.07	
	5.24	3.54	0.72	3.54	—	
1.0	1,435	157	157	100	157	103
		0.04	0.08	7.35	22.67	352.04
		10.63	5.04	0.48	5.04	—
	2,435	172	172	96	172	102
		0.04	0.08	8.39	21.57	251.83
		8.93	6.31	0.91	6.31	—
3,435	158	158	100	158	105	
	0.04	0.08	7.53	23.69	49.08	
	5.91	4.03	0.53	4.03	—	

Table A.34: Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	366	366	313	366	316
		0.16	0.31	58.60	7.17	125.67
	2,383	0.12	0.09	0.003	0.09	—
		352	352	295	352	296
	3,373	0.14	0.28	55.16	8.99	120.60
		0.14	0.12	0.01	0.12	—
0.75	1,580	349	349	309	349	313
		0.15	0.32	55.87	7.81	166.32
	2,580	0.16	0.08	0.01	0.08	—
		491	491	355	491	362
	3,576	0.37	0.32	79.38	25.37	198.89
		462	462	0.02	0.32	—
*1.0	1,780	0.14	0.28	81.11	33.06	352
		0.35	0.30	0.01	0.30	197.17
	2,780	483	483	367	483	369
		0.15	0.30	78.45	27.52	246.74
	3,780	0.28	0.24	0.01	0.24	—
		613	613	373	613	Out of Memory
*1.0	1,780	0.14	0.28	96.67	45.66	325.77
		0.72	0.64	0.08	0.64	—
	2,780	604	604	370	604	Out of Memory
		0.12	0.26	93.67	44.63	298.90
	3,780	0.76	0.67	0.07	0.67	—
		618	618	385	618	Out of Memory
3,780	0.12	0.26	96.87	42.82	301.46	
	0.60	0.54	0.05	0.54	—	

Table A.35: Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	305	305	245	305	246
		0.14	0.28	47.12	24.29	134.51
		0.43	0.33	0.001	0.33	—
	2,383	301	301	241	301	243
		0.14	0.29	45.83	24.98	322.92
		0.43	0.35	0.02	0.35	—
3,373	301	301	234	301	238	
	0.19	0.34	46.68	22.34	117.59	
	0.42	0.32	0.01	0.32	—	
0.75	1,580	375	375	262	375	272
		0.14	0.29	58.71	57.88	213.79
		0.87	0.72	0.06	0.72	—
	2,580	365	365	255	365	263
		0.15	0.30	55.83	60.0	672.32
		0.83	0.69	0.04	0.69	—
3,576	359	359	269	359	278	
	0.14	0.28	56.21	61.48	704.02	
	0.69	0.51	0.02	0.51	—	
*1.0	1,780	442	442	257	442	Out of Memory
		0.13	0.28	69.07	89.28	307.21
		1.48	1.27	0.11	1.27	—
	2,780	439	439	266	439	Out of Memory
		0.14	0.28	67.74	89.06	313.94
		1.37	1.17	0.12	1.17	—
3,780	436	436	267	436	Out of Memory	
	0.14	0.28	68.26	90.30	309.79	
	1.38	1.15	0.17	1.15	—	

Table A.36: Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 100\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	205	205	149	205	151
		0.15	0.28	29.47	50.87	427.90
		3.01	1.90	0.13	1.90	—
	2,383	208	208	141	208	149
		0.15	0.28	29.48	47.34	126.49
		2.77	1.92	0.22	1.92	—
0.75	3,373	194	194	139	194	142
		0.16	0.29	27.95	48.90	361.56
		2.52	1.57	0.15	1.57	—
	1,580	242	242	146	242	154
		0.16	0.30	35.39	93.12	1287.57
		6.87	4.01	0.35	4.01	—
*1.0	2,580	227	227	152	227	155
		0.14	0.27	32.73	95.36	1307.09
		3.80	2.64	0.34	2.64	—
	3,576	251	251	156	251	156
		0.14	0.28	36.87	90.14	1103.88
		6.98	4.20	0.83	4.20	—
*1.0	1,780	258	258	177	258	Out of Memory
		0.15	0.29	38.27	142.82	455.46
		8.94	5.38	1.74	5.38	—
	2,780	241	241	151	241	Out of Memory
		0.13	0.26	34.97	147.23	576.27
		7.19	3.97	0.99	3.97	—
3,780	255	255	168	255	Out of Memory	
	0.15	0.29	37.75	135.96	717.44	
	8.64	4.76	1.33	4.76	—	

Table A.37: Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 1, Demand $\in \{1, 10\}$

Density	Instances, $edge\#$	One-Step-2 $edge\#$ $time(sec.)$ $\%gap$	Reroute $edge\#$ $time(sec.)$ $\%gap$	Edge Del. $edge\#$ $time(sec.)$ $\%gap$	Edge Add. $edge\#$ $time(sec.)$ $\%gap$	Optimal Sol. $edge\#$ $time(sec.)$ ---
0.5	1,212	172	172	116	172	121
		0.04	0.08	7.29	3.63	16.43
		1.95	1.53	0.17	1.53	—
	2,201	165	165	115	165	120
		0.04	0.08	7.12	3.23	46.98
		1.59	1.22	0.13	1.22	—
	3,204	170	170	115	170	122
		0.04	0.08	6.91	3.37	15.34
		1.85	1.35	0.06	1.35	—
0.75	1,320	211	211	128	211	136
		0.06	0.11	10.04	9.86	33.91
		3.44	2.35	0.10	2.35	—
	2,309	198	198	123	198	134
		0.04	0.08	8.72	9.85	72.50
		3.12	2.42	0.24	2.42	—
	3,316	217	217	130	217	137
		0.04	0.09	9.83	8.87	87.23
		3.17	2.41	0.13	2.41	—
1.0	1,435	254	254	124	254	136
		0.04	0.09	11.48	15.09	172.89
		5.23	4.27	0.57	4.27	—
	2,435	247	247	118	247	128
		0.03	0.08	10.81	15.64	53.27
		6.03	4.83	0.51	4.83	—
	3,435	266	266	131	266	144
		0.04	0.08	12.10	14.0	179.90
		5.63	4.54	0.48	4.54	—

Table A.38: Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 10, Demand $\in \{1, 10\}$

Density	Instances, $edge\#$	One-Step-2 $edge\#$ $time(sec.)$ $\%gap$	Reroute $edge\#$ $time(sec.)$ $\%gap$	Edge Del. $edge\#$ $time(sec.)$ $\%gap$	Edge Add. $edge\#$ $time(sec.)$ $\%gap$	Optimal Sol. $edge\#$ $time(sec.)$ ---
0.5	1,212	135	135	84	135	87
		0.04	0.09	5.14	6.76	48.75
		4.81	3.40	0.45	3.40	—
	2,201	132	132	88	132	93
		0.04	0.08	5.92	5.99	16.44
		3.87	3.09	0.51	3.09	—
	3,204	131	131	85	131	88
		0.04	0.08	5.12	6.31	15.82
		4.06	3.01	0.14	3.01	—
0.75	1,320	154	154	92	154	97
		0.04	0.09	6.28	14.62	115.05
		6.89	4.84	0.44	4.84	—
	2,309	141	141	90	141	98
		0.04	0.08	6.07	14.54	188.58
		4.89	3.60	0.64	3.60	—
	3,316	158	158	97	158	98
		0.04	0.09	6.51	13.84	31.47
		6.45	4.83	0.12	4.83	—
1.0	1,435	165	165	93	165	95
		0.04	0.08	5.93	22.20	58.68
		11.01	6.99	1.19	6.99	—
	2,435	186	186	90	186	93
		0.04	0.08	7.14	20.51	3604.11
		13.74	10.39	1.97	10.39	—
	3,435	177	177	95	177	102
		0.03	0.08	7.26	20.95	74.14
		9.41	7.32	1.85	7.32	—

Table A.39: Computational Results of the One-Step-2 Algorithm, Node # = 30, Coefficient = 100, Demand $\in \{1, 10\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,212	90	90	52	90	51
		0.04	0.08	3.18	10.36	226.68
		19.88	16.91	2.33	16.91	—
	2,201	95	95	50	95	55
		0.06	0.10	3.58	8.97	3601.96
		20.17	17.72	2.54	17.72	—
	3,204	89	89	54	89	58
		0.04	0.08	3.29	9.61	640.36
		13.41	11.11	2.17	11.11	—
0.75	1,320	107	107	52	107	57
		0.04	0.08	4.28	18.96	3603.20
		28.91	24.41	2.83	24.41	—
	2,309	112	112	49	112	55
		0.06	0.10	4.82	18.34	3616.24
		36.69	32.09	3.18	32.09	—
	3,316	103	103	55	103	57
		0.04	0.08	4.80	18.07	3602.83
		19.59	17.13	2.91	17.13	—
1.0	1,435	143	143	52	143	57
		0.04	0.08	5.04	23.92	3603.82
		73.88	62.88	4.48	62.88	—
	2,435	137	137	53	137	57
		0.04	0.08	4.62	24.25	3603.76
		61.71	54.21	4.31	54.21	—
	3,435	119	119	55	119	58
		0.04	0.08	4.47	25.18	3603.80
		33.24	28.58	3.58	28.58	—

Table A.40: Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 1, Demand $\in \{1, 10\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	275	275	183	275	184
		0.14	0.28	40.51	32.43	127.19
		1.95	1.43	0.10	1.43	—
	2,383	266	266	179	266	185
		0.14	0.28	39.24	32.47	126.10
		1.97	1.48	0.08	1.48	—
	3,373	276	276	185	276	194
		0.14	0.28	40.12	27.36	122.41
		1.70	1.31	0.07	1.31	—
0.75	1,580	345	345	179	345	193
		0.14	0.29	51.86	66.17	872.01
		4.34	3.22	0.26	3.22	—
	2,580	342	342	170	342	194
		0.13	0.28	49.93	66.07	1297.85
		4.53	3.46	0.45	3.46	—
	3,576	346	346	196	346	206
		0.14	0.29	52.34	70.17	854.12
		3.08	2.34	0.28	2.34	—
*1.0	1,780	419	419	186	419	Out of Memory
		0.14	0.29	64.10	95.54	359.60
		6.32	5.04	0.88	5.04	—
	2,780	402	402	180	402	Out of Memory
		0.13	0.28	59.22	98.68	102.05
		5.76	4.69	0.97	4.69	—
	3,780	441	441	185	441	Out of Memory
		0.14	0.28	67.29	89.35	721.88
		6.60	5.54	1.05	5.54	—

Table A.41: Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 10, Demand $\in \{1, 10\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	1,390	207	207	133	207	144
		0.14	0.27	28.21	50.33	711.18
		4.42	3.01	0.35	3.01	—
	2,383	206	206	127	206	132
		0.14	0.27	27.61	47.74	147.80
		4.34	3.16	0.19	3.16	—
	3,373	202	202	131	202	135
		0.14	0.27	27.85	46.78	448.13
		3.55	2.65	0.19	2.65	—
0.75	1,580	241	241	139	241	145
		0.14	0.28	33.52	65.51	1066.10
		7.31	5.19	0.65	5.19	—
	2,580	242	242	130	242	134
		0.17	0.30	33.09	91.82	3609.27
		7.97	5.99	0.83	5.99	—
	3,576	250	250	137	250	147
		0.14	0.28	35.26	89.76	3188.68
		7.59	5.37	0.76	5.37	—
*1.0	1,780	290	290	137	290	Out of Memory
		0.12	0.26	41.53	127.74	1088.13
		10.48	8.27	1.18	8.27	—
	2,780	262	262	129	262	Out of Memory
		0.12	0.25	35.72	131.40	2693.73
		9.37	6.19	1.22	6.19	—
	3,780	284	284	134	284	Out of Memory
		0.12	0.26	40.16	129.0	1778.69
		10.77	8.45	1.14	8.45	—

Table A.42: Computational Results of the One-Step-2 Algorithm, Node # = 40, Coefficient = 100, Demand $\in \{1, 10\}$

Density	Instances, edge#	One-Step-2 edge# time(sec.) %gap	Reroute edge# time(sec.) %gap	Edge Del. edge# time(sec.) %gap	Edge Add. edge# time(sec.) %gap	Optimal Sol. edge# time(sec.) ---
0.5	*1,390	143	143	78	143	RLE
		0.14	0.27	17.32	66.53	3607.56
		18.01	15.90	5.13	15.90	—
	2,383	147	147	103	147	84
		0.14	0.26	16.12	64.76	3606.05
		21.82	18.01	6.22	18.01	—
3,373	141	141	75	141	81	
	0.14	0.26	17.58	62.05	3606.06	
	18.66	15.85	4.12	15.85	—	
*0.75	1,580	189	189	74	189	RLE
		0.14	0.27	21.84	106.01	3607.26
		26.64	19.91	5.46	19.91	—
	2,580	171	171	79	171	RLE
		0.15	0.28	24.06	108.27	3607.94
		25.79	22.14	6.89	22.14	—
3,576	194	194	85	194	RLE	
	0.14	0.27	22.16	104.61	3607.18	
	27.83	22.63	7.14	22.63	—	
*1.0	1,780	212	212	83	212	RLE
		0.12	0.26	27.38	145.60	3609.82
		48.32	40.07	8.01	40.07	—
	2,780	178	178	83	178	RLE
		0.12	0.25	24.50	149.43	3609.79
		37.25	30.88	7.65	30.88	—
3,780	212	212	85	212	RLE	
	0.12	0.26	28.87	145.05	3609.75	
	57.58	51.71	9.48	51.71	—	