

ALGORITHMS FOR 2 EDGE CONNECTIVITY WITH FIXED COSTS IN TELECOMMUNICATIONS NETWORKS

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY

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

By
Umut Güzel
July, 2011

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 Karařan(Advisor)

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.

Assist. Prof. Ali Aydın Selçuk

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School of Engineering and Science

ABSTRACT

ALGORITHMS FOR 2 EDGE CONNECTIVITY WITH FIXED COSTS IN TELECOMMUNICATIONS NETWORKS

Umut Güzel

M.S. in Industrial Engineering

Supervisor: Assoc. Prof. Dr. Oya Karaşan

July, 2011

In this thesis, several algorithms are developed in order to provide cost-effective and survivable communication in telecommunications networks. In its broadest sense, a survivable network is one which can maintain communication even in the presence of a physical breakdown. There are several ways of providing survivable communication in a given network. Our choice is to hedge against single link failures and provide two edge disjoint paths for every source and destination pair. Each edge in the network is assumed to have a variable unit routing cost and a fixed usage cost. Our objective is the minimization of the total routing cost of the traffic demand and the fixed cost of the utilized links. Several constructive and improvement type heuristics are developed and tested extensively in an experimental design setting.

Keywords: 2 edge connectivity in telecommunications networks, Survivable networks, Primal and secondary paths .

ÖZET

HABERLEŞME AĞLARINDA SABİT MALİYETLİ 2 AYRIT BAĞLILIK İÇİN ALGORİTMALAR

Umut Güzel

Endüstri Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Doç. Dr. Oya Karahan

Temmuz, 2011

Bu tez kapsamında, haberleşme ağlarında uygun maliyetli ve güvenilir iletişim sağlamak amacıyla çeşitli algoritmalar tasarlanmıştır. En geniş tanım olarak güvenilir ağlar, fiziksel bir arıza oluştuğunda bile iletişimi sürdürebilen ağlardır. Haberleşme ağlarında güvenilirliği sağlamanın çeşitli yolları vardır. Tek bir ayrit arızasına karşı önlem almak amacıyla her bir kaynak hedef ikilisi için 2-ayrit yol buluyoruz. Ağdaki her bir ayrit rotalama maliyeti ve sabit maliyete sahiptir. Amacımız kullanılan her bir ayritın sabit giderleri ve talep trafiği rotalama maliyetleri toplamını enküçültmektir. Çeşitli sezgisel algoritmalar geliştirilmiş ve test edilmiştir.

Anahtar sözcükler: Haberleşme ağlarında 2 ayrit bağıllık, Güvenilir ağ tasarımı, Birincil ve ikincil yollar.

To my parents, my brother Utku and my dearest Emre...

Acknowledgement

I would like to express my sincere gratitude to both of my professors, Assoc. Prof. Dr. Oya Karařan and Assoc. Prof. Dr. Hande Yaman for their guidance, suggestions and supports during my graduate study.

I am indebted to Assist. Prof. Ali Aydın Selçuk for accepting to read and review this thesis and also for his valuable time and comments.

I am grateful to TÜBİTAK for their financial support during my research.

I would like to thank to my family; my mother and father, Fatma Jale Güzel and Bahaddin Güzel and my brother Utku Güzel for their endless love and support throughout all my education life.

I would like to express my deepest gratitude to my beloved boyfriend Emre İlbars for giving me encouragement and motivation and also for his amazing patience and support.

I am most thankful to Aşlı Deniz Güven and Utku Çulha for their morale support and making this whole process endurable by their presence in the university.

Finally, I would like to thank to all my colleagues from Bilkent and friends for their support and friendship.

Contents

1	Introduction	1
2	Literature Review	4
2.1	Uncapacitated Fixed Charge Network Design Literature	5
2.2	Disjoint Paths Literature	6
2.3	Heuristics Literature	9
3	Mathematical Model Development	12
3.1	Fixed Charge Single Path Network Design Problem (SPND) . . .	13
3.2	Fixed Charge 2 Edge-Disjoint Paths Network Design Problem (2EDPND)	15
4	Heuristic Algorithms	17
4.1	Route Algorithm	18
4.2	RouteAll Algorithm	22
4.3	One-Step Algorithm	23
4.4	Moves	24

4.4.1	Add Edge	25
4.4.2	Delete Edge	25
4.4.3	Cycle Edge	26
4.5	Improvement Heuristics: Local Search	26
4.5.1	First Selection Algorithms	27
4.5.2	Best Selection Algorithms	29
4.5.3	Best Of Three	30
4.6	Improvement Heuristics: Basic Variable Neighborhood De- scent(VND)	30
4.6.1	Priority1	32
4.6.2	Priority2	32
4.7	Improvement Heuristics: General Variable Neighborhood Descent	35
4.7.1	VNS1	35
4.7.2	VNS2	36
5	Computational Results	39
5.1	Test Instances	39
5.2	Results of the Tests	40
5.2.1	Results for the Problem of Fixed Charge Single Path Net- work Design Problem (SPND)	42
5.2.2	Results for the Problem of Fixed Charge 2 Edge-Disjoint Paths Network Design Problem (2EDPND)	45

<i>CONTENTS</i>	ix
6 Conclusion	51
A Computational Results	55

List of Figures

4.1	An example to show how Suurballe's algorithm work	20
5.1	Instance-Gap Graphs for SPND Problem	44
5.2	Instance-Gap Graphs for 2EDPND Problem	47
5.3	Results of VNS algorithms for 2EDPND Problem for instances with $ V = 30$	49
5.4	Results of VNS algorithms for 2EDPND Problem for instances with $ V = 40$	50

List of Tables

3.1	Notation	13
5.1	Average results for SPND Problem	43
5.2	Average results for 2EDPND Problem	46
A.1	Computational Results for SPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,10\}$	56
A.2	Computational Results for SPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,100\}$	57
A.3	Computational Results for SPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,10\}$	58
A.4	Computational Results for SPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,100\}$	59
A.5	Computational Results for SPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,10\}$	60
A.6	Computational Results for SPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,100\}$	61
A.7	Computational Results for SPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,10\}$	62

A.8	Computational Results for SPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,100\}$	63
A.9	Computational Results for SPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,10\}$	64
A.10	Computational Results for SPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,100\}$	65
A.11	Computational Results for SPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,10\}$	66
A.12	Computational Results for SPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,100\}$	67
A.13	Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,10\}$	68
A.14	Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,100\}$	69
A.15	Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,10\}$	70
A.16	Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,100\}$	71
A.17	Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,10\}$	72
A.18	Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,100\}$	73
A.19	Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,10\}$	74

A.20 Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,100\}$	75
A.21 Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,10\}$	76
A.22 Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,100\}$	77
A.23 Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,10\}$	78
A.24 Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,100\}$	79

Chapter 1

Introduction

Networks are being used in many fields today. With the fast development of the Internet and broadened scope of communication, usage of networks has become wide spread and the requirements of the networks increased. A telecommunications network consists of links and nodes. Links enable transferring data or signal between the nodes of the networks. Nodes can represent several different devices in a network such as switches or electrical devices. Links may represent wires, electrical cables, radio waves or optical fibers. There exist many different types of telecommunications networks and each one of them has different purposes. Telephone networks communicate in the form of sound, optical networks transfer data in the form of light, electrical networks transfer data in the form of electricity etc. Within the scope of this thesis is any one of these telecommunications networks where a survivable communication is essential.

Efficiency and reliability are the basic requirements in a network. In order to provide reliability, a network has to be capable of recovering failures and accidents that occur in nodes or links or any other device in the network. Networks have to be survivable to continue communication or data transfer in case of a failure. In a survivable network design there should exist more than one alternative way to provide communication; for example, there could be more than one path between a source and a destination pair. These additional paths are referred to as the recovery paths (secondary paths) for each communicating pair. Under normal

circumstances the primary or the initially designated path is used, but when a node or link failure occurs in the primary path of a pair, then the secondary path of this pair is used until the failure in the primary path is repaired. These paths need to be link or node disjoint. In this thesis, as is commonly done in the literature, it is assumed that only one failure can typically occur at a time in a network. Thus choosing edge or node disjoint paths between communication pairs will hedge against such breakdowns. In our thesis, edge disjointness will be adopted.

In the literature, survivability problem in networks is studied with different objectives like minimizing the total routing costs, minimizing the cost of the shorter path, minimizing the cost of the longer path or setting bounds for the costs of the paths. The objective of the problem might render the problem to be NP-hard. Furthermore, cost structure for the networks can also vary. In a network, cost for the primary path c_p and the secondary path c_s could be either identical or different. If these two costs are different, there may be a relation between them as $c_s = \alpha c_p$ such that $0 < \alpha < 1$ or conversely these costs may be arbitrary.

We are given a network $G = (N, E, C, f)$, in which N is the node set, E is the set of edges, C is the cost vector and f is the fixed cost vector. When an edge in E is used for the first time, a fixed cost is charged for installing that edge. However, if the same edge is used after it is installed, then only the routing cost for that edge is charged. Throughout the thesis, it is assumed that every possible node pair will communicate.

In this thesis we study two different problems; Fixed Charge Single Path Network Design Problem (SPND) and Fixed Charge 2 Edge-Disjoint Paths Network Design Problem (2EDPND). The first problem is to find a path for each communicating pair, while minimizing the total cost. Total cost includes the fixed costs for the installed edges and the routing costs of the paths. This problem is studied by many different researches through the literature. It is known to be NP-hard. The second problem is to find 2 edge-disjoint paths for each source and target node pair, while minimizing the total cost. The first path is called as the

primary path and the second path is called as the *secondary path*. We consider different cost structures for the unit routing costs of the paths. Unit routing cost of an edge for the primary path is equal to the original cost of this edge. Since the secondary paths are less frequently used, only in case of failures, we assume that the unit routing cost of an edge is halved if it is used on secondary path. In our problem, the relationship between the path costs is $c_s = 1/2c_p$. This problem, which considers different cost structures, has been studied in the literature. However, most of these problems do not consider fixed costs. This problem is also NP-hard.

In order to solve the latter problem, we construct heuristic algorithms. First of all, we use a construction algorithm to find an initial solution for our problem. Then, we construct improvement heuristics to enhance the initial solution. Different approaches are used while designing these improvement algorithms such as local search, variable neighborhood descent and variable neighborhood search. We also modified these algorithms to solve the former problem. Finally, these algorithms are tested and compared for both of the problems.

The remaining chapters of this thesis are organized as follows. Chapter 2 reviews the literature for the explained problems. Chapter 3 gives the mathematical models for the studied problems. In Chapter 4, construction and improvement heuristics are explained in detail. Chapter 5 states the computational analysis for the heuristics and mathematical models. In Chapter 6, thesis is summarized. In Appendix, tables of the numerical results can be found.

Chapter 2

Literature Review

Survivability has become a basic necessity for the network problems arising in telecommunications applications. In a survivable network, backup or recovery path must be available in addition to the primary path for each pair in case of a failure. If a failure occurs in a survivable network, that network will be able to recover itself. In normal circumstances primary path is used, but if a link or node failure happens in the primary path of a pair, then that pair will establish connection using the backup path. Since the backup path is used when the primary path becomes unserved, these two paths must be edge or node disjoint.

We first focus on the problem of finding a single path for every possible node pair in a given network. If a single routing cost is used for each edge in the problem, then the problem is easy. It is enough to find the shortest paths to solve this problem. However, when fixed costs for each edge are considered the problem becomes more complicated. This problem is referred as the uncapacitated fixed charge network design problem in the literature. Literature review for *uncapacitated fixed charge network design* can be found in the first section.

Secondly, we work on the problem of finding two edge-disjoint paths for every potential source and target node pair in a given network G considering the routing costs and fixed costs, while minimizing the total cost. For this problem, unit

routing cost of an edge for primary path is equal to the original cost of the edge and unit routing cost of the same edge for secondary path is equal to the half of the original cost of that edge. In both of the problems total cost includes routing cost of the paths for each commodity and sum of the fixed costs of utilized edges. When the literature is searched, it is seen that most of the problems in disjoint paths literature do not consider fixed costs. Literature review for *disjoint paths* can be found in the second section.

Both of the explained problems are NP-hard. Even though both problems are NP-hard, designing heuristic algorithms is more challenging for the second one. We have written a construction heuristic to find an initial feasible solution. But, finding an initial solution is not enough. In order to find better solutions, we construct several improvement algorithms which are applied to the initial solution. In the last section, literature review for *heuristics* can be found.

2.1 Uncapacitated Fixed Charge Network Design Literature

Holmberg et al. [11] propose an exact algorithm to solve the problem of uncapacitated fixed charge network design (UFCND). This algorithm uses Lagrangian heuristic with a branch and bound framework as a basis. This heuristic consists of three main parts; a construction heuristic to find a good initial solution, a subgradient search procedure and the Lagrangian relaxation of the problem.

Cruz et al. [5] also illustrates an exact algorithm which combines the branch and bound method and Lagrangian relaxation. They suggest a new technique to choose the branching variables for the branch and bound method.

Duhamel [7] states that UFCND is NP-hard. Duhamel illustrates two metaheuristics to solve this problem. These metaheuristics are tested on the instances with up to 400 nodes and fixed cost of an edge is calculated as ten times the routing cost. For smaller instances, optimal value is found. However as the instances

become larger running time increases, as a result the algorithm cannot approach to the optimal solution.

2.2 Disjoint Paths Literature

Through the literature, there exists several different versions of the problem of finding disjoint paths. Mainly, the objective functions of the problems are different. Some problems aim to minimize the total cost whereas others aim to minimize the cost of a specific path. Furthermore, some problems search for edge disjointness, whereas the others search for node disjointness.

Damci [6] studies the problem of finding 2 edge disjoint paths between every possible source and target node in a given network. Additional to the routing costs, fixed costs for the edges are considered in this thesis. Objective is to minimize the total cost. Two construction algorithms and four different improvement methods are illustrated and tested for large number of instances. This problem is the same as our second problem, namely 2EDPND. We use the One-Step version 2 algorithm of Damci and we use the proposed improvement methods while we are constructing our improvement heuristics to solve our problem.

Suurballe [17] describes an algorithm to find k node-disjoint paths between a single pair of source and sink nodes, with minimum total length. The given algorithm finds the desired k paths at k iterations, at each iteration a single shortest path algorithm is used. As it is stated in [18], this algorithm can also be modified to find the k edge-disjoint paths. This modification of the algorithm is an $O(m \log_{1+m/n} n)$ -time algorithm and it uses $O(m)$ space, where m is the number of edges and n is the number of nodes. For each pair, Suurballe and Tarjan [18] uses two iterations of Dijkstra's single source algorithm. This algorithm solves the explained problem with optimality. In Suurballe's problem, each edge is assigned only one cost whereas in our problem three different cost values are assigned to a single edge. For each edge $\{i, j\}$: unit routing cost for primary path is c_{ij}^1 , unit routing cost for secondary path is c_{ij}^2 and the fixed cost for opening is f_{ij} .

Lee et al. [12] state an algorithm for the problem of finding k -best paths. It is the problem of finding k paths which are as different as possible and have the minimum cost in total. The algorithm described finds k paths which are both node-disjoint and edge-disjoint if there exist such paths, if not it finds k paths such that they are edge-disjoint and have minimum number of nodes in common. On the contrary, in our problem we want to find edge-disjoint paths between pairs. These paths can have common nodes since we have no restriction for node sharing.

Li et al. [13] consider the problem of finding k disjoint paths on a given graph G between s and t while minimizing the total cost of paths. This problem is different than those existing in the literature in that each edge has k different edge costs, where j^{th} cost of an edge is related to the j^{th} path. Four different versions of the stated problem are analyzed. Disjoint paths may be node or edge disjoint and the given graph may be directed or undirected. Li et al. show that all four variants of the problem is NP-complete even when $k=2$. They construct a polynomial time algorithm for the acyclic directed case and give heuristics for the proposed problem. In the heuristic algorithm, they average the costs on each edge and then solve a Minimum Cost Network Problem (MCNF). This study seems similar to our study, since they consider different edge costs for different paths. However, their costs have no relation, whereas we have a relation as: $c^2 = c^1/2$. In addition, they do not consider fixed costs for the active edges.

Xu et al. [19] work on the problem of finding two disjoint paths such that the cost of the cheaper path is minimized. It is called the *Min-Min* problem. They proved that this problem is NP-complete. A heuristic is given for the *Min-Min* problem in which a divide-and-conquer technique is used. Different from the *Min-Min* problem, *Min-Max* problem is also studied in [14]. *Min-Max* problem aims to minimize the cost of the path which is more expensive than the other. Both of these problems are different from our problem in the sense of their objectives; our objective is to minimize the total cost which includes the costs of the primary and secondary paths for each commodity and plus the sum of the fixed costs of active edges.

Zheng et al. [20] discuss the problem of finding minimum-cost k paths subject to defined minimum link/node sharability constraints. Large node or link sharability on a set of paths means, paths in this set have more nodes or links in common. If link/node sharability for a path set is 0, then the paths in this set are link/node disjoint. The authors define five main sharability constraints: min-sum node sharability, min-sum link sharability, min-max node sharability, min-max link sharability and the last one is having no restriction for the sharability. They describe sixty five different constraints which are composite of the five main constraints defined. An algorithm is given for the general problem. They proved that this problem for twenty five of the composite constraints are polynomially solvable. This study is fundamentally different from ours, we want two paths to be only link-disjoint and we do not impose any restrictions on the node disjointness.

Bhatia et al. [2] focus on the problem of finding a pair of disjoint paths of minimum total cost where cost of an edge for primary path is c_{ij} and cost of the same edge for backup path is αc_{ij} , where $\alpha < 1$. As a result, the contribution of the cost of the backup path is smaller than the contribution of the cost of the primary path. They give an algorithm for the stated problem having an approximation ratio of $O(1/\alpha)$. This problem is very similar to ours, since in our problem $\alpha=1/2$. However, as it is commonly done in the literature, the authors do not consider fixed costs for the activated edges. They state that if α is fixed, like in our study, NP-hardness of the problem is still an open question.

Gomes et al. [8] study the problem of finding k disjoint paths from a specified source node to target node, in a network such that each edge has k different cost values. Their objective is to minimize the total cost. They prove that, the proposed problem is NP-complete even when $k=2$. The authors give an exact algorithm to the problem of finding two arc disjoint paths for every node pair in a given network. This algorithm can be modified to find two node disjoint paths if desired. Similarly, with slight changes path lengths can also be bounded. The authors, work on the instances with at most 1000 nodes. Since each edge has k different cost values, this study is very similar to the problem discussed by Li et al. [13], which is previously stated. Our problem seems to be a special case of this problem for $k=2$, however each edge has two different costs in both problems, in

our problem these two costs have a relation between them whereas their costs are arbitrary. In addition, like the previous papers fixed costs of the edges are not considered.

Perl et al. [16] work on the problem of finding two disjoint paths P_1 and P_2 such that P_1 is from s_1 to t_1 and P_2 is from s_2 to t_2 . They consider four variants of the problem; the given network can be directed or undirected, paths can be edge or node disjoint. They provide relations among these four different versions and construct efficient algorithms for some of the cases. This study is fundamentally different from our study, we find a pair of disjoint paths between a source and a target node whereas they find a pair of disjoint paths between two different pairs of nodes.

2.3 Heuristics Literature

Heuristic algorithms find good feasible solutions in reasonable computation times. We are interested in two different types of heuristics: construction heuristic and improvement heuristic. For a given problem, a construction heuristic finds an initial feasible solution and an improvement heuristic starts from an initial solution and searches for a better solution. An example of an improvement heuristic is a local search algorithm. It starts with an initial solution and searches among a subset of solutions in a given neighborhood until the local optimum is found.

Sometimes local optima can be far away from the global optimal solution. In these cases meta-heuristics are needed. Metaheuristics are used when classic heuristics and combinatorial methods fail to solve complex optimization problems [15]. Metaheuristics have become an important application area through the last two decades. Metaheuristics aim to escape from the local optima found by local searches by for instance letting worsening solutions [3].

Variable neighborhood search (VNS) basically uses the idea of the systematic change of the neighborhoods as it is stated in [9] by Hansen et al. By the help of the neighborhood change VNS finds the local optima and then escapes from

the valleys that include these local optima. Hansen et al. explains the basic components of the Variable Neighborhood Search (VNS) and present some extensions of the VNS in their paper. Variable Neighborhood Descent (VND) is the basic version of VNS. VND algorithm starts with an initial solution as the current solution, assigns the first neighborhood as the current neighborhood and finds the best neighbor of the current solution in the current neighborhood. If the new solution is better than the current solution, current solution is updated and first neighborhood is assigned to the current neighborhood. If not, the next neighborhood is assigned as the current neighborhood and the algorithm continues. VND terminates when no improvement exist.

Extensions of VNS are given such that: Reduced Variable Neighborhood Search (RVNS), Basic Variable Neighborhood Search (BVNS) and General Variable Neighborhood Search (GVNS). RVNS is very similar to the VND algorithm. The only difference is that, VND selects the best neighbor in the current neighborhood, whereas RVNS selects a random neighbor from the current neighborhood. BVNS, like RVNS generates a random neighbor in the current neighborhood but instead of comparing this point with the current solution, it applies a local search to the randomized point compare the new solution obtained from the local search with the current solution. GVNS starts with an existing solution x which is found by applying a local search to an initial solution found by a construction heuristic. After that, starting from the first neighborhood, a random solution is generated in the current neighborhood and VND is applied to this solution and obtain x' . If the cost of x' is better than the cost of x , then x is updated as: $x \leftarrow x'$ and the algorithm continues with the first neighborhood. If not the next neighborhood is selected.

Variable neighborhood search algorithm is used to solve several different problems in the literature. Bräysy [4] modifies the variable neighborhood descent algorithm to solve the problem of vehicle routing with time windows. Instead of using the original algorithm, a few changes are done. First of all, instead of finding the best neighbor, first neighbor in the current neighborhood is found. Furthermore, they changed the parameter values and objective function. The

algorithm changes the objective function to escape from the local optima. Hemmelmayer et al. [10] gives a new heuristic to solve the periodic routing problems without time windows. The authors use VNS as the underlying method of their heuristic.

In our improvement algorithms, we first use the idea of local search. Then we implement two different variants of Variable Neighborhood Search; Variable Neighborhood Descent and General Variable Neighborhood Search.

In the next chapter, our problems will be formalized and mathematical models will be provided.

Chapter 3

Mathematical Model Development

In this chapter integer programs for the studied problems are given. We study two problems, the first one is the Fixed Charge Single Path Network Design Problem (SPND) and the second one is the Fixed Charge 2 Edge-Disjoint Paths Network Design Problem (2EDPND).

We are given a network $G = (N, E, C, f)$, where N is the node set, E is the edge set, C is the cost vector and f is the fixed cost vector. Both directions of a link can be utilized for communication. Thus, for the given network we create an arc set A as: $\{(i, j) \cup (j, i) : \{i, j\} \in E\}$. Additionally, a commodity set K corresponding to a set of pairs of nodes in N is given. For each commodity $k \in K$, $s_k \in N$ is the source node, $t_k \in N$ is the destination node and $d_k \in Z_+$ is the demand between source and destination. Each edge $\{i, j\} \in E$ has two assigned costs, one is the unit routing cost C_{ij} and the second one is the fixed cost f_{ij} , which will be paid when an edge is used for the first time only. Notation can be seen at Table 3.1.

In the first problem, namely SPND, our aim is to find a single path for each commodity, while minimizing the total cost. Total cost consists of the fixed costs of active edges and the routing costs on the paths found for each commodity

Table 3.1: Notation

$G = (N, E, C, f)$	: Given network
N	: Set of nodes in G
E	: Set of edges in G
C	: Cost vector of G
f	: Fixed cost vector of G
K	: Set of commodities in G
c_{ij}^1	: Unit routing cost for primary path of edge $\{i, j\} \in E$
c_{ij}^2	: Unit routing cost for secondary path of edge $\{i, j\} \in E$
s_k	: Source node of commodity $k \in K$
t_k	: Target node of commodity $k \in K$
d_k	: Demand of commodity $k \in K$

$k \in K$. Let $\mathcal{P}_k$ correspond to the single path for commodity k . For ease of notation, we shall sometimes view $\mathcal{P}_k$ as the collection of edges or arcs on this path. Cost of this path is calculated as follows, $\text{cost}(\mathcal{P}_k) = \sum_{\{i,j\} \in E \cap \mathcal{P}_k} d_k C_{ij}$. Total cost is defined as: $\sum_{\{i,j\} \in E: (i,j) \text{ or } (j,i) \text{ appears on some } \mathcal{P}_k} f_{ij} + \sum_{k \in K} \text{cost}(\mathcal{P}_k)$.

In the second problem, namely 2EDPND, our aim is to find two edge-disjoint paths for every commodity, while minimizing the total cost. Each edge in our network has two different routing costs; c_{ij}^1 is the unit routing cost for the primary path $c_{ij}^1 = C_{ij}$ and c_{ij}^2 is the unit routing cost for the secondary path $c_{ij}^2 = C_{ij}/2$. Let $\mathcal{P}_k, \mathcal{S}_k$ be the primary and secondary path for commodity k . $\text{cost}(\mathcal{P}_k) = \sum_{\{i,j\} \in E \cap \mathcal{P}_k} d_k c_{ij}^1$ and $\text{cost}(\mathcal{S}_k) = \sum_{\{i,j\} \in E \cap \mathcal{S}_k} d_k c_{ij}^2$. In this problem, the total cost consists of the fixed costs of active edges and the sum of routing costs of the two paths found for each commodity $k \in K$. Total cost is defined as: $\sum_{\{i,j\} \in E: (i,j) \text{ or } (j,i) \text{ appears on some } \mathcal{P}_k \text{ or } \mathcal{S}_k} f_{ij} + \sum_{k \in K} \text{cost}(\mathcal{P}_k) + \text{cost}(\mathcal{S}_k)$.

3.1 Fixed Charge Single Path Network Design Problem (SPND)

In this section mathematical model of the first problem will be provided. In the model two decision variables are used. Variables are as follows: binary variable x_{ijk} indicates the arcs utilized on the path of commodity k and binary variable

y_{ij} indicates the set of active edges.

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

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

Model for SPND:

$$\min \sum_{\{i,j\} \in E} f_{ij} y_{ij} + \sum_{k \in K} \sum_{\{i,j\} \in E} C_{ij} (x_{ijk} + x_{jik}) d_k \quad (3.1.1)$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in A} x_{ijk} - \sum_{j:(j,i) \in A} x_{jik} = \begin{cases} 1, & \text{if } i = s_k \\ -1, & \text{if } i = t_k \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in N, \forall k \in K \quad (3.1.2)$$

$$x_{ijk} + x_{jik} \leq y_{ij} \quad \forall \{i, j\} \in E, \forall k \in K \quad (3.1.3)$$

$$x_{ijk} \in \{0, 1\} \quad \forall (i, j) \in E, \forall k \in K \quad (3.1.4)$$

$$y_{ij} \in \{0, 1\} \quad \forall \{i, j\} \in E \quad (3.1.5)$$

The objective function (3.1.1) consists of two parts; the first part is the summation of the fixed costs of the activated edges and the second one is the summation of the routing costs of a single path for each commodity $k \in K$. Constraint (3.1.2) is the flow balance constraint required for each commodity. Constraint (3.1.3) guarantees that if an edge $\{i, j\} \in E$ is used on the path of a commodity, that edge is activated.

3.2 Fixed Charge 2 Edge-Disjoint Paths Network Design Problem (2EDPND)

In this section mathematical model of the second problem will be provided. In the model three decision variables are used. Variables are as follows: binary variable xp_{ijk} indicates the arcs utilized on the primary path of commodity k , binary variable xs_{ijk} indicates the arcs utilized on the secondary path of commodity k and binary variable y_{ij} keeps track of active edges.

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

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

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

Model for 2EDPND:

$$\min \sum_{\{i,j\} \in E} f_{ij} y_{ij} + \sum_{k \in K} \sum_{\{i,j\} \in E} (c_{ij}^1(xp_{ijk} + xp_{jik}) + c_{ij}^2(xs_{ijk} + xs_{jik})) d_k \quad (3.2.1)$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in A} xp_{ijk} - \sum_{j:(j,i) \in A} xp_{jik} = \begin{cases} 1, & \text{if } i = s_k \\ -1, & \text{if } i = t_k \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in N, \forall k \in K \quad (3.2.2)$$

$$\sum_{j:(i,j) \in A} xs_{ijk} - \sum_{j:(j,i) \in A} xs_{jik} = \begin{cases} 1, & \text{if } i = s_k \\ -1, & \text{if } i = t_k \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in N, \forall k \in K \quad (3.2.3)$$

$$xp_{ijk} + xs_{ijk} + xp_{jik} + xs_{jik} \leq y_{ij} \quad \forall \{i, j\} \in E, \forall k \in K \quad (3.2.4)$$

$$xp_{ijk}, xs_{ijk} \in \{0, 1\} \quad \forall (i, j) \in A, \forall k \in K \quad (3.2.5)$$

$$y_{ij} \in \{0, 1\} \quad \forall \{i, j\} \in E \quad (3.2.6)$$

The objective function of the model (3.2.1) consists of two parts; the first part is the summation of the fixed costs of the active edges and the second one is the summation of the routing costs of both paths for each commodity $k \in K$. In the model there are two flow balance constraints (3.2.2) and (3.2.3). The first one is for the primary path and the second one is used for the secondary path for each commodity $k \in K$. Constraint (3.2.4) forces that an edge $\{i, j\} \in E$ is used on at most one path; either primary or secondary for each commodity $k \in K$. It also ensures that if an edge $\{i, j\} \in E$ is used on a path of a commodity, that edge is activated.

In the next chapter, heuristic algorithms are given in detail.

Chapter 4

Heuristic Algorithms

In this chapter, we detail our heuristic algorithms. These heuristics are designed to solve the problem of Fixed Charge 2 Edge-Disjoint Paths Network Design (2EDPND). Later on we modify these heuristics to be able to solve Fixed Charge Single Path Network Design (SPND) problem.

We use heuristics to get close to the optimal solution as much as we can. First of all, we obtain an initial feasible solution by a construction heuristic. Starting from this initial solution, several improvement heuristics using different methods are used to search and find better solutions. Improvement heuristics depend on three main heuristics, two of which are metaheuristics. As we know local search gets stuck at a local optimum. However, metaheuristics let algorithm to escape from the local optima. The first part of the improvement heuristics use the method of local search, the second part of the heuristics use the basic variable neighborhood descent and the last part of the heuristics use the general variable neighborhood search.

In our study, we have a network $G=(N, E, C, f)$ in which N denotes the node set, E denotes the edge set, C denotes the cost vector and f denotes the fixed cost vector. Each edge $\{i, j\} \in E$ has a fixed cost f_{ij} in addition to a unit routing cost C_{ij} . The aim of our problem is to find, two edge disjoint paths for every possible node pair, while minimizing the cost in total. A commodity set

K is also given, which includes all possible node pairs as commodities. For each commodity $k \in K$, s_k denotes the source node, t_k denotes the destination node and d_k denotes the demand requested between source and destination nodes. We shall use $\mathcal{P}$ as a path as well as the collection of edges or arcs present in $\mathcal{P}$.

As we have seen in the *disjoint paths* literature, for the problem of finding two edge disjoint paths between a single pair of source node s and destination node t , Suurballe's algorithm [18] is a well known algorithm which solves the problem with optimality. This algorithm constitutes the basic idea of our heuristics.

In the first section, Route algorithm is illustrated. Route algorithm mainly uses Suurballe's algorithm to find two edge-disjoint paths between the source and destination nodes of a given commodity $k \in K$. In the second section, RouteAll algorithm is given. RouteAll finds two edge disjoint paths for all commodities in K . Whenever the active edge set of a solution is updated, all commodities are rerouted using the RouteAll algorithm. In the third section, a construction algorithm is described which finds an initial feasible solution for the problem. Our construction algorithm is named as One-Step Algorithm. In the fourth section, our three main moves which are used in the improvement algorithms; adding an edge, deleting an edge and cycling an edge are defined. In the fifth section, improvement algorithms using the idea of local search are described. In the sixth section, improvement algorithms using the idea of basic variable neighborhood descent are illustrated. Finally in the last section, improvement algorithms using the idea of general variable neighborhood are explained.

4.1 Route Algorithm

Route algorithm finds two edge disjoint paths between two nodes of the given network, using the algorithm of Suurballe [18]. In Suurballe's problem, each edge $\{i, j\} \in E$ has only one routing cost C_{ij} . The objective is to minimize the total routing cost of these two paths. This algorithm is an $O(m \log_{1+m/n} n)$ -time algorithm and it uses $O(m)$ space, where n is the number of nodes and m is the

number of edges. This algorithm is a modification of the algorithm given in [17], which finds two node-disjoint paths between a single pair.

Suurballe's algorithm aims to find two edge disjoint paths between a source node s and a target node t . First of all, it finds the shortest path tree rooted at source node s using the Dijkstra's algorithm [1], then constructs the first path to node t from this shortest path tree. Original cost for an edge $\{i, j\} \in E$ is C_{ij} and shortest path distance from s to $u \in N$ is denoted as d_{su} . Updated cost of an edge $\{i, j\} \in E$ is $C'_{ij} = C_{ij} + d_{si} - d_{sj}$. Directions of the edges on the first shortest path are reversed. The second shortest path is found between s and t using the updated costs and edge directions. Finally, the common edges in the first and second paths are discarded and two edge-disjoint paths from s to t are constructed. For a deeper understanding of this algorithm, Figure 4.1 can be analyzed. In this example our aim is to find 2 edge-disjoint paths between node 1 and node 3. Number on the edges are the edge costs. Part (a) of the figure shows the original network. In part (b), first path is constructed using the Dijkstra's algorithm. Part (c) shows the transformed network. In part (d) using Dijkstra for the second time, the second path is constructed. Finally, after discarding the common edges, 2 edge disjoint paths are found between nodes 1 and 3. Path P represents the primary path and Path S represents the secondary path.

Route algorithm takes a network $G'=(N', E', C')$, in which N' denotes the node set, E' denotes the edge set, C' denotes the cost vector and a commodity k as an input and finds two edge-disjoint paths from s_k to t_k in G' . First of all, a directed graph has been constructed from G' . An arc set $\vec{E}$ is defined for the directed graph. For each edge $\{i, j\} \in E'$, arcs (i, j) and $(j, i) \in \vec{E}$. Costs of arcs (i, j) and (j, i) are equal to C'_{ij} . It finds the paths using the Suurballe's algorithm and assigns the path with the smaller cost as the primary path and the path with the larger cost as the secondary path and calculates the total cost for the commodity k . This algorithm outputs the primary and secondary path and the cost for the given commodity k : $\mathcal{P}_k$, $\mathcal{S}_k$ and $cost_k$. The pseudo code of Route algorithm can be found in Algorithm 1.

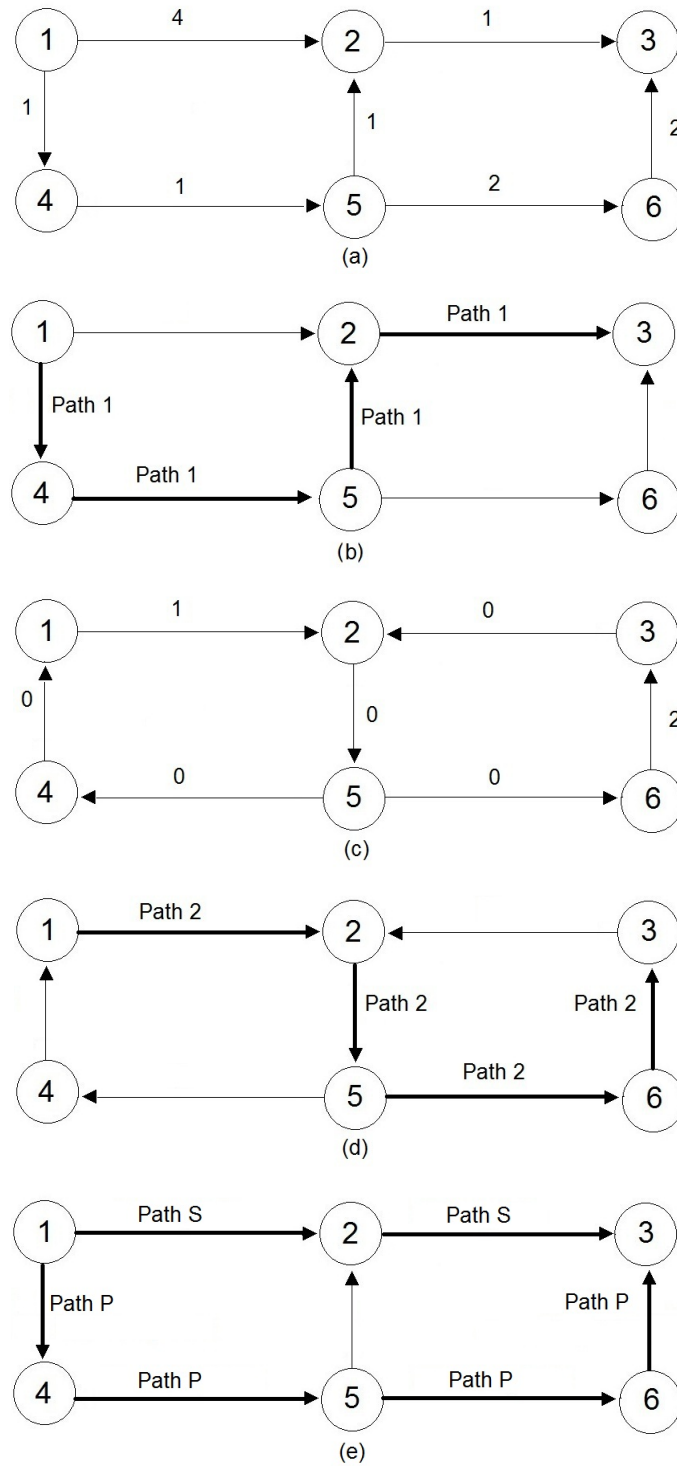


Figure 4.1: An example to show how Suurballe's algorithm work

Algorithm 1 Route

INPUT: a network $G'=(N',E',C', f)$, a commodity k

OUTPUT: primary path $\mathcal{P}$, secondary path $\mathcal{S}$, $cost$

Let $\vec{E} = \{(i,j) \cup (j,i) : \{i,j\} \in E'\}$, $\vec{C}_{ij} = \vec{C}_{ji} = C'_{ij} \ \forall \{i,j\} \in E'$ and $\vec{G}=(N,\vec{E},\vec{C})$

Compute a shortest path tree rooted at node s_k in graph $\vec{G}$ using Dijkstra's algorithm.

Let P denote a shortest path from node s_k to node t_k .

Let $d_{s_k u}$ denote the shortest path distance from node s_k to node u .

Let $\vec{G}' = (N', \vec{E}', \vec{C}')$ where $\vec{E}'$ is the same as $\vec{E}$ except that the directions of arcs on path P are reversed and $\forall (i,j) \in \vec{E}': \vec{C}'_{ij} = \vec{C}_{ij} + d_{s_k i} - d_{s_k j}$

Compute a shortest path P' from node s_k to node t_k in $\vec{G}'$ using Dijkstra's algorithm.

if one or more links appear in both P and P' (in opposite direction) **then**

Discard these links.

Construct 2 edge disjoint paths from s_k to t_k using remaining links.

Let $\mathcal{P} \leftarrow$ path with the smaller cost

Let $\mathcal{S} \leftarrow$ path with the larger cost

$cost = (\sum_{\{i,j\} \in \mathcal{P}} C_{ij} + \sum_{\{i,j\} \in \mathcal{S}} C_{ij}/2)d_k$

4.2 RouteAll Algorithm

RouteAll algorithm constructs two edge-disjoint paths for each commodity $k \in K$. This algorithm takes a network G' as an input and using the Route algorithm k times, it finds two edge-disjoint paths between s_k and t_k for every commodity $k \in K$ in G' . It returns the solution found as an output. For a solution X , X_{A-} denotes the set of active edges, X_R denotes the routing set, which includes two paths $\mathcal{P}_k$ and $\mathcal{S}_k$ for all $k \in K$, X^w denotes the weight matrix which keeps track of the demand carried by each edge $\{i, j\} \in E$ and $X_{totalcost}$ denotes the total cost. In order to know the amount of demand that is carried by a single edge, weight of that edge is calculated for every edge $\{i, j\} \in E$. Weight of an edge $\{i, j\}$ in a solution X is denoted as X_{ij}^w and equals to $\sum_{k \in K: \{i, j\} \in \mathcal{P}_k} d_k + \sum_{k \in K: \{i, j\} \in \mathcal{S}_k} d_k/2$. Weight of an edge is used in two different steps; in one step algorithm to distribute fixed cost of the edges and in cycle move. *Totalcost* of a solution X is calculated as: $\sum_{k \in K} cost_k + \sum_{\{i, j\} \in X_{A-}} f_{ij}$. The first part is the sum of the costs of the commodities, and the second part is the sum of the fixed costs of the active edges. Pseudo code of the given algorithm can be seen in Algorithm 2.

Algorithm 2 RouteAll

INPUT: a network $G'=(N', E', C', f)$

OUTPUT: a solution X

$X_{A-} \leftarrow \emptyset$

$X_R \leftarrow \emptyset$

for all k **in** K **do**

 Route(G', k) and get $\mathcal{P}_k, \mathcal{S}_k, cost_k$

 Add paths $\mathcal{P}_k$ and $\mathcal{S}_k$ to set X_R

$X_{A-} \leftarrow X_{A-} \cup \mathcal{P}_k \cup \mathcal{S}_k$

$X_{ij}^w = \sum_{k \in K: \{i, j\} \in \mathcal{P}_k} d_k + \sum_{k \in K: \{i, j\} \in \mathcal{S}_k} d_k/2$ for all $\{i, j\} \in X_{A-}$

$X_{totalcost} = \sum_{k \in K} cost_k + \sum_{\{i, j\} \in X_{A-}} f_{ij}$

4.3 One-Step Algorithm

One-Step algorithm is a construction heuristic, which is used to find an initial feasible solution for our problem. In One-Step algorithm, all commodities in K are routed three different times using Suurballe's algorithm. Since Suurballe uses a single routing cost for each edge $\{i, j\} \in E$, while solving our problem it is important to decide which cost structure to use for edges.

As an initial step, a routing is done with only using the original routing costs of the edges C , without considering the fixed costs. Each commodity $k \in K$ is considered separately in this case. Then, the set K is sorted in the descending order of costs of commodities and second application of the routing is done in this order. Assume, solution X is obtained from the first routing.

In the second application, we use the fixed costs of the edges additional to their original unit routing costs. We force algorithm to consider fixed costs of the edges when assigning the paths. For the solution X , which is found at the end of first application, we calculate a weight X_{ij}^w for each edge $\{i, j\} \in E$. We use weight to distribute the fixed costs of the edges. Before applying RouteAll, we update cost of each edge $\{i, j\} \in E$ as $C_{ij} + f_{ij}/X_{ij}^w$. During the second application, after each commodity is routed, cost of $\{i, j\} \in E$ which have been activated is updated as C_{ij} . At the end of this application, we obtain a new solution X .

In the third and last application of RouteAll, original unit routing costs are used. This time we use only the activate edges in X_{A-} . The logic behind applying the RouteAll algorithm for the third time is to close the unused edges. At the end of this algorithm, an initial feasible solution X is returned. Pseudo code of One-Step algorithm can be found in Algorithm 3.

Algorithm 3 One-Step

 INPUT: original network $G=(N,E,C,f)$

 OUTPUT: an initial solution X

 RouteAll(G) and get solution X

 Sort the set K in descending order of $cost_k$
 $C'_{ij} \leftarrow C_{ij} + f_{ij}/X_{ij}^w$ for all $\{i,j\} \in E$

 Let $G' = (N, E, C')$

 Define a new solution X'
 $X'_{A^-} \leftarrow \emptyset$
for all $k \in K$ **do**

 Route(G', k) and get $\mathcal{P}_k, \mathcal{S}_k, cost_k$

 $X'_{A^-} \leftarrow X'_{A^-} \cup \mathcal{P}_k \cup \mathcal{S}_k$

 $C'_{i,j} \leftarrow C_{i,j}$ for all $\{i,j\} \in (\mathcal{P}_k \cup \mathcal{S}_k)$

 Let $G^- = (N, X'_{A^-}, C)$

 RouteAll(G^-) and get solution X

4.4 Moves

Until now, we only focused on constructing an initial feasible solution. From this point on we are interested in improving the initial solution. Our aim is to find a better solution than the initial one. As we explained before, each solution has a set of active edges. Two main modifications can be done in the active edge set. The first one is activating an inactive edge; rerouting with the new active edge set leads us to a new solution. Since, we activate a new edge, some previous edges can become unused and as a result of this, the new solution can have a smaller total cost than the previous one. On the other hand, if none of the previous edges becomes inactive, total cost can be larger than the previous one. The second one is inactivating an active edge in this set; rerouting with the new active edge set leads us to a new solution. This solution may be infeasible. If the new solution is feasible, then the cost of the new solution is compared with the current one. It can be either smaller or larger.

We define three different moves considering these two main modifications. These moves are as follows: *Add Edge*, *Delete Edge* and *Cycle Edge*. They will be used inside the improvement algorithms. *Cycle Edge* uses the idea of both *Add*

Edge and *Delete Edge*. For a solution X , for each move, an edge set is defined as $X_{Movename}$ which includes the potential edges which lead feasible solutions when this move is applied to X .

4.4.1 Add Edge

For every solution X , X_{Add} is defined. X_{Add} includes all the inactive edges in the solution X . *Add Edge* move takes an inactive edge $\{i, j\} \in X_{Add}$ and a solution X as input. The given edge is activated and added to the set of active edges of the given solution. All commodities are rerouted using the new set of active edges and original routing costs, at the end a new solution is found. Pseudo code of Add move can be seen in Algorithm 4.

Algorithm 4 Add Edge

INPUT: $\{i, j\}$ and a solution X

OUTPUT: a solution X

Let $G^- = (N, (X_A^- \cup \{i, j\}), C)$

RouteAll (G^-) and get solution X

4.4.2 Delete Edge

This move is the opposite of the previous one. For every solution X , X_{Delete} is defined. X_{Delete} includes all the active edges in the solution X such that inactivation of that edge leads to a feasible solution. *Delete Edge* move takes an active edge $\{i, j\} \in X_{Delete}$ and a solution X as input. The given edge is inactivated, i.e., removed from the set of active edges of the given solution. All commodities are rerouted using the new set of active edges and original routing costs, at the end a new solution is found. Pseudo code of this move can be seen in Algorithm 5.

Algorithm 5 Delete Edge

INPUT: $\{i, j\}$ and a solution X OUTPUT: a solution X Let $G^- = (N, (X_{A^-} \setminus \{i, j\}), C)$ RouteAll (G^-) and get solution X

4.4.3 Cycle Edge

We construct a move which includes the two main modifications: adding an edge and deleting an edge. This move is the composition of the previous two moves. In this move an edge is inactivated while several edges are being activated. This move also takes an active edge and a solution X as input. This move has a main difference from the previous ones, it considers the fixed costs. Assume we are cycling the edge $\{i, j\} \in E$. First of all, cost vector is updated such that if an edge $\{k, l\} \in E$ is in the set of active edges of X its cost is C_{kl} , if not its cost is $C_{kl} + (f_{kl}/X_{ij}^w)$. For the given edge, a third path which has no common edges with the primary path and the secondary path is found using Dijkstra with the updated cost vector, if it exists. For every solution X , X_{Cycle} is defined. X_{Cycle} includes all the active edges in the solution X such that a third path can be found as it is described. The given active edge is inactivated, while the edges on the third path are activated and added to the set of active edges. All commodities are rerouted using the new set of active edges and original routing costs, at the end a new solution is found. Pseudo code of this move can be seen in Algorithm 6.

4.5 Improvement Heuristics: Local Search

In this part, improvement algorithms, which use the idea of local search, are presented. All of these algorithms are applied after running the One-Step construction heuristic. An initial feasible solution is found at the end of One-Step algorithm. These algorithms start from this initial solution as a current solution and search for new solutions. When new solutions are found with improved

Algorithm 6 Cycle Edge

INPUT: $\{i, j\}$ and a solution X OUTPUT: a solution X **for all** $\{k, l\} \in E$ **do** **if** $\{k, l\} \in A^-$ **then** $C'_{kl} \leftarrow C_{kl}$ **else** $C'_{kl} \leftarrow C_{kl} + f_{kl}/X_{ij}^w$ Temporarily delete the edges used in primary and secondary paths of $\{i, j\}$ pair and find a new shortest path $P^=$ from i to j using Dijkstra's algorithm with the cost vector C' Let $G^- = (N, (X_{A^-} \setminus \{i, j\} \cup P^=), C)$ RouteAll (G^-) and get solution X

costs, the current solution is updated. All improvement algorithms take the initial solution found by One Step algorithm as an input and output the current solution when the algorithm terminates. Local search heuristic, searches a given neighborhood of the initial solution. It stops at a local optima.

4.5.1 First Selection Algorithms

There exist three different versions of this algorithm, corresponding to the three different moves described above. These algorithms assign the initial solution as the current solution. In these algorithms, for the selected move $M \in \{Add, Delete, Cycle\}$ each edge $\{i, j\} \in X_M$ is selected one by one, move M is applied to X and new solutions are obtained. Among these new solutions, the first improving solution is selected and assigned as the current solution. This algorithm continues until none of the possible edges improve the current solution. Pseudo codes of the three different versions of the algorithm can be seen in Algorithms 7, 8 and 9.

Algorithm 7 Add First

INPUT: a solution X obtained from One-Step Algorithm
 OUTPUT: a solution X'

Let X' be the current solution
 $X' \leftarrow X$

repeat

Find an edge $\{i, j\} \in X'_{Add}$ that Add Edge $(\{i, j\}, X')$ returns a solution X
 such that $X_{totalcost} < X'_{totalcost}$

Update X' : $X' \leftarrow X$

until there does not exist such an edge $\{i, j\}$

Algorithm 8 Delete First

INPUT: a solution X obtained from One-Step Algorithm
 OUTPUT: a solution X'

Let X' be the current solution
 $X' \leftarrow X$

repeat

Find an edge $\{i, j\} \in X'_{Delete}$ that Delete Edge $(\{i, j\}, X')$ returns a solution
 X such that $X_{totalcost} < X'_{totalcost}$

Update X' : $X' \leftarrow X$

until there does not exist such an edge $\{i, j\}$

Algorithm 9 Cycle First

INPUT: a solution X obtained from One-Step Algorithm
 OUTPUT: a solution X'

Let X' be the current solution
 $X' \leftarrow X$

repeat

Find an edge $\{i, j\} \in X'_{Cycle}$ that Cycle Edge $(\{i, j\}, X')$ returns a solution
 X such that $X_{totalcost} < X'_{totalcost}$

Update X' : $X' \leftarrow X$

until there does not exist such an edge $\{i, j\}$

4.5.2 Best Selection Algorithms

There exist also three different versions of this algorithm, corresponding to the three different moves described above. In these algorithms, for the selected move $M \in \{Add, Delete, Cycle\}$ each edge $\{i, j\} \in X_M$ is selected, move M is applied one by one to X and new solutions are obtained. The solution for the edge $\{i, j\} \in X_M$, that has the smallest total cost is selected and if it improves the current solution, it is assigned as the current solution, if not the algorithm terminates. Pseudo codes of the three different versions of the algorithm can be seen in Algorithms 10, 11 and 12.

Algorithm 10 Add Best

INPUT: a solution X obtained from One-Step Algorithm

OUTPUT: a solution X'

Let X' be the current solution

$X' \leftarrow X$

repeat

Find an edge $\{i, j\} \in X'_{Add}$ such that Add Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost

if cost is improved **then**

Update X' : $X' \leftarrow X$

until cost is not improved

Algorithm 11 Delete Best

INPUT: a solution X obtained from One-Step Algorithm

OUTPUT: a solution X'

Let X' be the current solution

$X' \leftarrow X$

repeat

Find an edge $\{i, j\} \in X'_{Delete}$ such that Delete Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost

if cost is improved **then**

Update X' : $X' \leftarrow X$

until cost is not improved

Algorithm 12 Cycle Best

INPUT: a solution X obtained from One-Step AlgorithmOUTPUT: a solution X' Let X' be the current solution $X' \leftarrow X$ **repeat**Find an edge $\{i, j\} \in X'_{Cycle}$ such that Cycle Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost**if** cost is improved **then**Update X' : $X' \leftarrow X$ **until** cost is not improved

4.5.3 Best Of Three

In this algorithm our aim is to select the most improving solution found among the best selections of the three moves: Add, Cycle and Delete for a current solution. As an initial step, Best Of Three takes the initial solution as the current solution X . Then for the current solution checks for the edge $\{i, j\} \in X_{Add}$ which has the solution X with the smallest total cost, checks for the edge $\{i, j\} \in X_{Delete}$ which has the solution X with the smallest total cost and lastly checks for the edge $\{i, j\} \in X_{Cycle}$ which has the solution X with the smallest total cost. Among these three edges, the one finds the solution with the smallest total cost is selected and if it improves the current cost, then the current solution is updated. If not, the algorithm terminates. Pseudo code of this algorithm can be seen in Algorithm 13.

4.6 Improvement Heuristics: Basic Variable Neighborhood Descent(VND)

In this section, improvement algorithms, which use the idea of the basic variable neighborhood search, are described. All of these algorithms are applied after running the One-Step construction heuristic. Both of these algorithms take the initial solution found by One Step algorithm as an input and output the current

Algorithm 13 Best Of Three

INPUT: a solution X obtained from One-Step AlgorithmOUTPUT: a solution X' $flag \leftarrow true$ $found \leftarrow false$ Let X' be the current solution and X'' be the temporary solution $X' \leftarrow X$ **while** $flag$ **do** $X' \leftarrow X$ Find an edge $\{i, j\} \in X'_{Add}$ that Add Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost**if** $X_{totalcost} < X''_{totalcost}$ **then**Update X'' : $X'' \leftarrow X$ $found \leftarrow true$ Find an edge $\{i, j\} \in X'_{Delete}$ that Delete Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost**if** $X_{totalcost} < X''_{totalcost}$ **then**Update X'' : $X'' \leftarrow X$ $found \leftarrow true$ Find an edge $\{i, j\} \in X'_{Cycle}$ that Cycle Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost**if** $X_{totalcost} < X''_{totalcost}$ **then**Update X'' : $X'' \leftarrow X$ $found \leftarrow true$ **if** $found$ **then**Update X' : $X' \leftarrow X$ **else** $flag \leftarrow false$

solution when the algorithm terminates. Variable neighborhood descent assigns the first neighborhood as the current neighborhood and finds the best neighbor of the current solution in the current neighborhood. If the new solution is better than the current solution, current solution is updated and first neighborhood is assigned as the current neighborhood. If not, the next neighborhood is assigned as the current neighborhood and the algorithm continues. VND terminates when no improvement exist. We have to order the three moves as the neighborhoods. Since *Cycle Edge* move includes the other two moves, we assigned it as the first neighborhood.

4.6.1 Priority1

The neighborhoods are ordered as: *Cycle Edge*, *Delete Edge*, *Add Edge*. As an initial step, Priority1 takes the initial solution as the current solution X . As the first step, for the solution X it finds the edge $\{i, j\} \in X_{Cycle}$ which returns the solution X' with the smallest total cost. If it improves the current cost, then algorithm updates the current solution and repeats previous step until the cost is not improved. If current cost is not improved, finds the edge $\{i, j\} \in X_{Delete}$ which returns the solution X' with the smallest total cost. If it improves the current cost, the algorithm updates the current solution and goes back to the first step. If cost is not improved, finds the edge $\{i, j\} \in X_{Add}$ which returns the solution X' with the smallest total cost. If it improves the current cost, the algorithm updates the current solution and goes back to the first step. If cost is not improved, the algorithm terminates. Pseudo code of this algorithm can be found in Algorithm 14.

4.6.2 Priority2

This algorithm is very similar to the previous one. The only difference is the order of the neighborhoods. The new order of the neighborhoods is: *Cycle Edge*, *Add Edge*, *Delete Edge*. Pseudo code of this algorithm can be found in Algorithm 15.

Algorithm 14 Priority1

INPUT: a solution X obtained from One-Step AlgorithmOUTPUT: a solution X' $flag \leftarrow true$ Let X' be the current solution $X' \leftarrow X$ **while** $flag$ **do** Find an edge $\{i, j\} \in X'_{Cycle}$ such that Cycle Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost **if** cost is improved **then** Update X' : $X' \leftarrow X$ **else** Find an edge $\{i, j\} \in X'_{Delete}$ such that Delete Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost **if** cost is improved **then** Update X' : $X' \leftarrow X$ **else** Find an edge $\{i, j\} \in X'_{Add}$ such that Add Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost **if** cost is improved **then** Update X' : $X' \leftarrow X$ **else** $flag \leftarrow false$

Algorithm 15 Priority2

INPUT: a solution X obtained from One-Step AlgorithmOUTPUT: a solution X' $flag \leftarrow true$ Let X' be the current solution $X' \leftarrow X$ **while** $flag$ **do** Find an edge $\{i, j\} \in X'_{Cycle}$ such that Cycle Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost **if** cost is improved **then** Update X' : $X' \leftarrow X$ **else** Find an edge $\{i, j\} \in X'_{Add}$ such that Add Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost **if** cost is improved **then** Update X' : $X' \leftarrow X$ **else** Find an edge $\{i, j\} \in X'_{Delete}$ such that Delete Edge $(\{i, j\}, X')$ returns a solution X with the smallest cost **if** cost is improved **then** Update X' : $X' \leftarrow X$ **else** $flag \leftarrow false$

4.7 Improvement Heuristics: General Variable Neighborhood Descent

In this section, improvement algorithms, which use the idea of the general variable neighborhood search, are explained. All of these algorithms are applied after running the One-Step construction heuristic. Both of these algorithms take the initial solution found by One Step algorithm as an input and output the current solution when the algorithm terminates. General VNS starts with a current solution X which is found by applying a local search to an initial solution. After that, starting from the first neighborhood, a random solution is generated in the current neighborhood and VND is applied to this solution and obtain X' . If the cost of X' is better than the cost of X , then X is updated as: $X \leftarrow X'$ and the algorithm continues with the first neighborhood. If not the next neighborhood is selected. Priority1 is used as the VND algorithm that is used in GVNS. We define a limitation for the number of edges randomly selected at each step of the algorithm, which is denoted as NI , to have a termination condition. This limitation is calculated as a percentage of the number of active edges for the solution X found by the algorithm Priority1. NI is equal to $|X_A| \cdot \alpha$. We assign two different values to α : 0,05 and 0,1. The following two algorithms have only one difference, it is the logic of the neighborhood change.

4.7.1 VNS1

One Step algorithm is performed and an initial solution is obtained. Then, the algorithm Priority1 is applied to the initial solution and solution X is obtained. In this algorithm, there exists three main phases. In the first phase our neighborhood is the cycle move. The algorithm selects an edge randomly from the set of X_{Cycle} . Cycle that edge and get a new solution, then apply Priority1 to the new solution. If the current solution is improved, update the current solution and try NI random edges for the new solution. If not, select a new random edge from the set X_{Cycle} . When NI random edges are tried and current solution is not improved, then the algorithm passes to the second phase. In the second phase our

neighborhood is the add move. The algorithm selects an edge randomly from the set of X_{Add} . Add that edge and get a new solution, then apply Priority1 to the new solution. If the current solution is improved, update the current solution and go back to the first phase. If not, select a new random edge from the set X_{Add} . When NI random edges are tried and current solution is not improved, then the algorithm passes to the third phase. In the third phase our neighborhood is the delete move. The algorithm selects an edge randomly from the set of X_{Delete} . Delete that edge and get a new solution, then apply Priority1 to the new solution. If the current solution is improved, update the current solution and go back to the first phase. If not, select a new random edge from the set X_{Delete} . When NI random edges are tried and current solution is not improved, the algorithm terminates. Pseudo code of this algorithm can be seen in Algorithm 16.

4.7.2 VNS2

VNS2 is very similar to the previous algorithm. The only difference is the logic behind the neighborhood change. The neighborhood for the first phase is the same; cycle. But when we pass to the second phase this time we enlarge the neighborhood set. Neighborhood set includes both cycle and add. The algorithm selects one of them randomly. When the algorithm starts to third phase, the neighborhood set is enlarged one more time. This time neighborhood set includes cycle, add and delete. The algorithm selects one of them randomly. Pseudo code of this algorithm is given in Algorithm 17.

Algorithm 16 VNS1

INPUT: a solution X obtained from One-Step AlgorithmOUTPUT: a solution X'' Let X'' be the current solution,Priority $v1(X)$ and get the solution X' , $X'' \leftarrow X'$ $NI \leftarrow \lceil |(X''_{A-})| * \alpha \rceil$ $flag \leftarrow true$ **while** $flag$ **do** $index \leftarrow 0$, $found \leftarrow false$ **while** $index < NI$ **do** Randomly select an edge $\{i, j\} \in X''_{Cycle}$, Cycle Edge($\{i, j\}, X''$) and get X Priority $v1(X)$ and get the solution X' **if** $X'_{totalcost} < X''_{totalcost}$ **then** $X'' \leftarrow X'$ $index \leftarrow 0$ **else** $index \leftarrow index + 1$ $index \leftarrow 0$ **while** ($found == false$ AND $index < NI$) **do** Randomly select an edge $\{i, j\} \in X''_{Add}$, Add Edge($\{i, j\}, X''$) and get X Priority $v1(X)$ and get the solution X' **if** $X'_{totalcost} < X''_{totalcost}$ **then** $X'' \leftarrow X'$ $found \leftarrow true$ **else** $index \leftarrow index + 1$ $index \leftarrow 0$ **while** ($found == false$ AND $index < NI$) **do** Randomly select an edge $\{i, j\} \in X''_{Delete}$, Delete Edge($\{i, j\}, X''$) and get X Priority $v1(X)$ and get the solution X' **if** $X'_{totalcost} < X''_{totalcost}$ **then** $X'' \leftarrow X'$ $found \leftarrow true$ **else** $index \leftarrow index + 1$ **if** $index == NI$ **then** $flag \leftarrow false$

Algorithm 17 VNS2

INPUT: a solution X obtained from One-Step Algorithm
 OUTPUT: a solution X''
 Let X'' be the current solution, $X'' \leftarrow X$
 Priority $v1(X)$ and get the solution X' , $X'' \leftarrow X'$
 $NI \leftarrow \lceil |(X''_{A-})| * \alpha \rceil$
 $flag \leftarrow true$
while $flag$ **do**
 $index \leftarrow 0$, $found \leftarrow false$
 while $index < NI$ **do**
 Randomly select an edge $\{i, j\} \in X''_{Cycle}$, Cycle Edge($\{i, j\}, X''$) and get X
 Priority $v1(X)$ and get the solution X'
 if $X'_{totalcost} < X''_{totalcost}$ **then**
 $X'' \leftarrow X'$, $index \leftarrow 0$
 else
 $index \leftarrow index + 1$
 $index \leftarrow 0$
 while ($found == false$ AND $index < NI$) **do**
 Randomly select an edge $\{i, j\} \in (X''_{Cycle} \cup X''_{Add})$
 if $\{i, j\} \in X''_{Cycle}$ **then**
 Cycle Edge($\{i, j\}, X''$) and get the solution X
 else
 Add Edge($\{i, j\}, X''$) and get the solution X
 Priority $v1(X)$ and get the solution X'
 if $X'_{totalcost} < X''_{totalcost}$ **then**
 $X'' \leftarrow X'$, $found \leftarrow true$
 else
 $index \leftarrow index + 1$
 $index \leftarrow 0$
 while ($found == false$ AND $index < NI$) **do**
 Randomly select an edge $\{i, j\} \in (X''_{Cycle} \cup X''_{Add} \cup X''_{Delete})$
 if $\{i, j\} \in X''_{Cycle}$ **then**
 Cycle Edge($\{i, j\}, X''$) and get the solution X
 else
 if $\{i, j\} \in X''_{Add}$ **then**
 Add Edge($\{i, j\}, X''$) and get the solution X
 else
 Delete Edge($\{i, j\}, X''$) and get the solution X
 Priority $v1(X)$ and get the solution X'
 if $X'_{totalcost} < X''_{totalcost}$ **then**
 $X'' \leftarrow X'$, $found \leftarrow true$
 else
 $index \leftarrow index + 1$
 if $index == NI$ **then**
 $flag \leftarrow false$

Chapter 5

Computational Results

In this chapter, construction and improvement algorithms, which are explained in the previous chapter are tested in a computer with 1.73 GHz Core i7 740 processor and 4 GB of RAM. In order to find the optimal solution for the instances, the mathematical model described in Chapter 4 are solved using GAMS 22.3 and CPLEX 10.1.0.

5.1 Test Instances

In this section construction of the test instances are explained.

The number of nodes ($|N|$) for the test instances is either 30 or 40. We could not take node number larger, because even when $|N| = 40$ in some instances our mathematical model runs out of memory. In such cases, when we do not have the optimal solution at hand, we use the lower bound obtained from the model to calculate *%gap*.

We have three different probability values $p = 1, 0.75, 0.5$. Edges in the network are generated randomly with a probability p . When $p = 1$, the network is complete.

Each node in the network is selected as a point from a grid of 100 x100. For an edge $\{i, j\} \in E$ cost of the edge is equal to C_{ij} . C_{ij} is calculated as twice of the Euclidean distance between the points i and j . Unit routing cost for the primary path is equal to c_{ij}^1 and unit routing cost for the secondary path is equal to c_{ij}^2 . These two costs are calculated as: $c_{ij}^1 = C_{ij}$ and $c_{ij}^2 = C_{ij}/2$ for each edge $\{i, j\} \in E$. Each edge has a fixed cost additionally. For each edge $\{i, j\} \in E$: $f_{ij} = coef \times C_{ij} + rn$, where rn is a random number such that $rn \in [0, 1000]$ and $coef$ is a coefficient such that $coef \in \{100, 1000, 10000\}$. We can use $coef$ control the relationship between the fixed cost and routing costs change.

Lastly, every possible node pair i and j such that $i \neq j$, is added as a commodity to the commodity set K . For each commodity $k \in K$ a demand value d_k is assigned. There exists two different ranges for d_k : $d_k \in [1, 10]$ or $d_k \in [1, 100]$.

5.2 Results of the Tests

In this section we will analyze the results we obtain after an extensive computational analysis. For the heuristics, $\%gap$ is calculated as: $(heuristic\ value - optimal\ value)/optimal\ value \times 100$. Whenever the mathematical model cannot find the optimal value, we use the lower bound it provides in place of the optimal value in the above formula. In these cases, if GAMS finds a current integer solution we also have a $\%gap$ for GAMS.

We test heuristics for the two different problems explained before; Fixed Charge Single Path Network Design Problem and Fixed Charge 2 Edge- Disjoint Paths Network Design Problem. The objective of these problems include both routing costs and fixed costs. The problem of finding a single path in a network with a single routing cost is an easy problem. Whenever, the fixed costs are introduced to this problem, the problem becomes more complicated.

First of all, we compare the improvement algorithms and point out the better ones. Secondly, we will compare heuristic results with the results of the mathematical model. For a detailed information, one can look at the tables in the

Appendix.

When we look at the results of all the improvement algorithms, we can conclude that, running times of the heuristics are affected by the characteristics of the test instances. When we increase node number from 30 to 40 or when number of edges is increased, the running times also increase accordingly. At first, we put a time bound of one hour for the GAMS, but then we extended it to ten hours. So, we calculate two *%gap* values for each heuristic; one for the one hour bound and the other for the ten hours bound.

When $|N|=30$ or $|N|=40$, as we increase the coefficient value, which is used for the fixed costs, the *%gap* increases accordingly. As a result of this, we can say that when routing costs and fixed costs are close to each other, when coefficient value is small, our problem becomes easier. The heuristics find solutions that are close to the optimal solutions, *%gap* for the heuristics are small. When these costs differ from each other, coefficient value is bigger, *%gap* values for the heuristics increase.

We have eleven improvement heuristics in total. Add First, Delete First, Cycle First, Add Best, Delete Best, Cycle Best, Best Of three, Priority1, Priority2, VNS1 and VNS2. The first six of them uses single moves, whereas the remaining five of them uses all three moves in several orders. When we compare the three algorithms; Add First, Delete First, Cycle First it is seen that Cycle First gives the best results. Similarly, when we compare algorithms; Add Best, Delete Best, Cycle Best we can easily conclude that Cycle Best gives the best results. Since VNS1 and VNS2 algorithms start by running Priority1, their *%gap* values are the same as or better than the *%gap* value of Priority1. When we consider all results together, we can conclude that last five heuristics are always better than the first six heuristics. From this point on, we will compare the last five improvement heuristics only.

5.2.1 Results for the Problem of Fixed Charge Single Path Network Design Problem (SPND)

The heuristics given in the previous chapter solves the problem of finding two edge disjoint paths for each commodity $k \in K$. We may easily modify them to solve the problem of finding a single path for each commodity. Then, we test these heuristics to solve this problem using the test instances.

In Table 5.1, the average results for the SPND problem of the five considered heuristics and the mathematical model are illustrated. For each row of the table there exist three instances with different density levels. We averaged the results of these instances. In addition to obtain results for VNS1, results of VNS1 with $\alpha = 0,05$ and $\alpha = 0,1$ are averaged. Same is applied for the VNS2 algorithm.

When we consider Figure 5.1 we can easily see that when coefficient is increased in the order of 100 or 1000 or 10000, $\%gap$ increases accordingly. For each coefficient value the $\%gap$ values for most of the considered heuristics are close to each other. When coefficient is 10000 the $\%gap$ values are nearly the same.

Another result we obtain is that, when we decrease demand value from $d \in [1, 100]$ to $d \in [1, 10]$, $\%gap$ of the improvement heuristics increase. Increasing the fixed costs and decreasing the demand values both affect the gaps in the same way.

When coefficient is 100 and $d \in [1, 100]$, mathematical model finds the optimal solutions of these instances. When we check the $\%gap$ values of the last five improvement algorithms, it is easily seen that all of them find nearly the same $\%gap$. When $d \in [1, 10]$ and $|N|=30$, all three instances are solved to optimality by GAMS. Two of them are solved in one hour, one of them is solved during the ten hours of time. Best Of Three and VNS2 find the closest solutions to the optimal. When $|N|=40$, GAMS could not solve the problems with optimality. But the $\%gap$ values are calculated. GAMS beats the heuristics in general, but heuristics have small running times compared to GAMS. VNS2 algorithm is the

Node Number	Coefficient	Demand	BestOf Three <i>time</i> <i>gap-1</i> <i>gap-10</i>	Priority 1 <i>time</i> <i>gap-1</i> <i>gap-10</i>	Priority 2 <i>time</i> <i>gap-1</i> <i>gap-10</i>	VNS1 <i>time</i> <i>gap-1</i> <i>gap-10</i>	VNS2 <i>time</i> <i>gap-1</i> <i>gap-10</i>	Model <i>time</i> <i>gap-1</i> <i>gap-10</i>
30	100	[1,10]	10,94 0,26 0,09	7,38 0,33 0,09	6,49 0,33 0,09	13,72 0,31 0,09	12,4 0,32 0,06	5892,7 0,19 0
30	100	[1,100]	17,32 0,05 —	12,41 0,04 —	10 0,04 —	32,23 0,03 —	32,71 0,03 —	59,53 0 —
30	1000	[1,10]	6,4 9,36 6,18	3,2 9,36 6,18	2,57 9,36 6,18	6,39 9,36 6,18	7,32 9,36 6,18	RLE 52,06 12,84
30	1000	[1,100]	11,74 0,19 0,4	8,46 0,44 0,24	7,19 0,44 0,24	16,07 0,44 0,24	14,49 0,31 0,24	2108,31 0,05 0
30	10000	[1,10]	7,04 29,88 29,65	2,34 29,95 29,73	2,2 29,95 29,73	6,43 29,88 29,65	7,4 29,81 29,58	RLE 97,09 72,13
30	10000	[1,100]	8,5 9,6 9,3	2,9 9,86 9,55	3,25 9,86 8,96	7,49 9,23 8,92	8,69 9,19 8,88	RLE 84,9 44,06
40	100	[1,10]	73,74 0,9 0,56	43,18 0,91 0,57	39,72 0,99 0,65	83,81 0,89 0,56	88,19 0,88 0,55	RLE 0,81 0,36
40	100	[1,100]	125,91 0,07 —	89,16 0,07 —	73,33 0,07 —	264,49 0,07 —	223,16 0,07 —	691,59 0 —
40	1000	[1,10]	39,72 12,49 9,33	15,08 10,97 7,84	14,56 10,97 7,84	44,76 10,45 7,34	45,68 10,07 6,96	RLE — —
40	1000	[1,100]	70,12 1,32 1,12	41,85 1,02 0,81	35,64 1,02 0,81	83,35 0,99 0,78	84,39 0,99 0,78	RLE 1,82 0,66
40	10000	[1,10]	36,36 45,79 29,46	5,73 46,33 29,96	5,42 49,33 29,96	20,54 45,79 29,46	26,74 45,79 29,46	RLE — 58,91
40	10000	[1,100]	51,81 10,12 3,08	13,61 9,62 2,6	13,33 9,62 2,6	38,26 9,29 2,3	46,14 9,36 2,6	RLE — 43,19

Table 5.1: Average results for SPND Problem

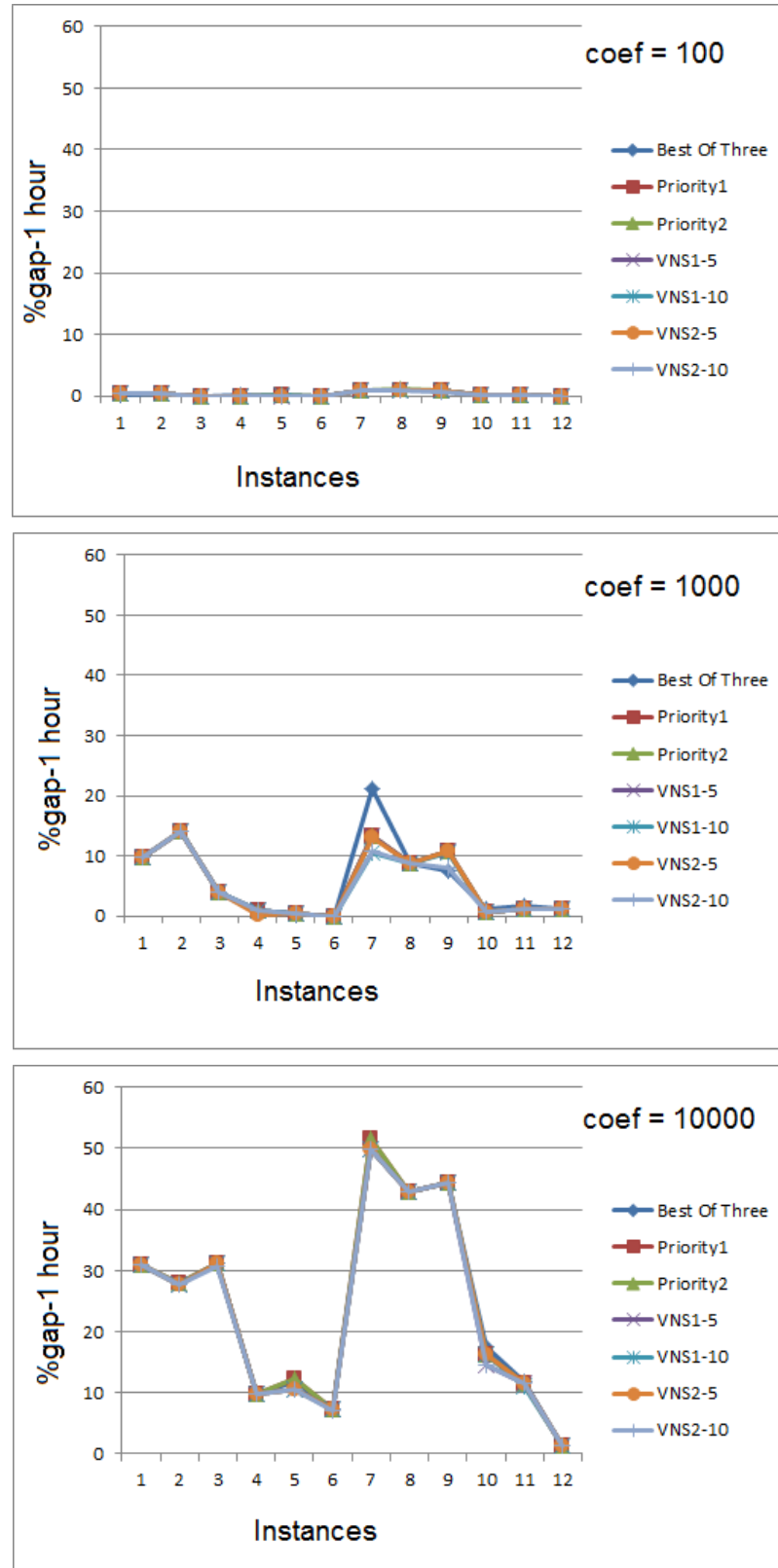


Figure 5.1: Instance-Gap Graphs for SPND Problem

one that obtain the best solutions in most cases.

When coefficient is 1000, $|N|=30$ and $d \in [1, 100]$ GAMS solves the instances to optimality. Two of the instances are solved in one hour and one of them is solved during the ten hours of time. All improvement algorithms we consider, find solutions which have nearly the same $\%gap$ s. When $d \in [1, 10]$ the mathematical model cannot find the optimal solution. When we compare the $\%gap$ values of heuristics and GAMS, heuristics beats GAMS. All heuristics considered, find similar gaps for each instance. When $|N|=40$ and $d \in [1, 10]$, GAMS cannot solve the problem with optimality, in addition to this, even in ten hours it cannot find any integer solution. VNS1 and Best Of Three find the closest solutions to the lower bound found by GAMS. When $|N|=40$ and $d \in [1, 100]$, GAMS cannot solve the problem to optimality, but for both one hour and ten hour $\%gap$ values are calculated. In one hour time limit, heuristics beat GAMS. However, in ten hours limit, GAMS beats heuristics. VNS and Priority algorithms found nearly the same gap values.

When coefficient is 10000, GAMS cannot find the optimal solutions even in ten hours. In very few cases $\%gap$ can be calculated and these gaps are larger than 85% when time limit is one hour. For the instances with $d \in [1, 10]$ $\%gap$ of the heuristics are larger than the $\%gap$ of the instances, in which $d \in [1, 10]$.

5.2.2 Results for the Problem of Fixed Charge 2 Edge-Disjoint Paths Network Design Problem (2EDPND)

In Table 5.2, the average results for the 2EDPND problem of the five considered heuristics and the mathematical model are illustrated.

When we consider Figure 5.2 we can easily see that when coefficient is increased in the order of 100 or 1000 or 10000, $\%gap$ increases accordingly. When coefficient is equal to 100, $\%gap$ values are close to each.

When we compare the two different demand values, we can easily see that, when $d \in [1, 10]$ the $\%gap$ for the heuristics increase.

Node Number	Coefficient	Demand	BestOf Three <i>time</i> <i>gap-1</i> <i>gap-10</i>	Priority 1 <i>time</i> <i>gap-1</i> <i>gap-10</i>	Priority 2 <i>time</i> <i>gap-1</i> <i>gap-10</i>	VNS1 <i>time</i> <i>gap-1</i> <i>gap-10</i>	VNS2 <i>time</i> <i>gap-1</i> <i>gap-10</i>	Model <i>time</i> <i>gap-1</i> <i>gap-10</i>
30	100	[1,10]	36,96 0,76 0,66	15,78 0,73 0,63	14,62 1,06 0,96	73,13 0,63 0,52	73,34 0,61 0,51	RLE 0,93 0,35
30	100	[1,100]	75,64 0,05 —	44,35 0,05 —	42,15 0,05 —	313,1 0,04 —	295,28 0,03 —	140,86 0 —
30	1000	[1,10]	46,6 4,43 4,34	10,84 5,57 5,48	12,26 5,8 5,71	38,76 4,1 4,02	40,17 4,04 3,95	RLE — 15,31
30	1000	[1,100]	54,38 0,84 0,67	30,87 0,84 0,67	31,87 0,82 0,66	76,54 0,76 0,59	88,26 0,68 0,51	RLE 3,59 0,23
30	10000	[1,10]	32,88 7,94 7,69	6,05 10,15 9,9	7,19 10,15 9,9	25,49 7,18 6,93	34,53 6,44 6,19	RLE — 23,76
30	10000	[1,100]	34,94 4,06 3,79	7,14 5,48 5,2	6,63 5,48 5,2	31,19 4,52 4,25	31,93 4,77 4,77	RLE — 14,52
40	100	[1,10]	254,63 0,79 0,51	212,57 0,79 0,5	184,44 0,77 0,5	1049,67 0,65 0,42	799,08 0,73 0,45	3000 0,94 0,35
40	100	[1,100]	705,88 0,02 —	368,93 0,02 —	322,61 0,02 —	1748,25 0,02 —	1432,79 0,02 —	1167,33 0 —
40	1000	[1,10]	258,36 7,75 4,52	47,04 7,69 2,46	115,09 7,23 4,02	265,83 7,24 4,03	268,09 7,08 3,88	RLE — —
40	1000	[1,100]	332,48 0,84 0,67	170,29 0,99 0,83	190,52 0,92 0,77	773,74 0,78 0,62	831,12 0,85 0,69	RLE 0,84 0,64
40	10000	[1,10]	272,49 10,08 8,16	26,35 13 10,86	39,21 13 10,86	159,75 10,95 8,84	236,73 10,54 8,43	RLE — —
40	10000	[1,100]	276,82 8,86 4,67	63,76 9,57 5,36	83,87 9,82 5,6	303,17 8,88 4,69	302,34 8,85 4,66	RLE — —

Table 5.2: Average results for 2EDPND Problem

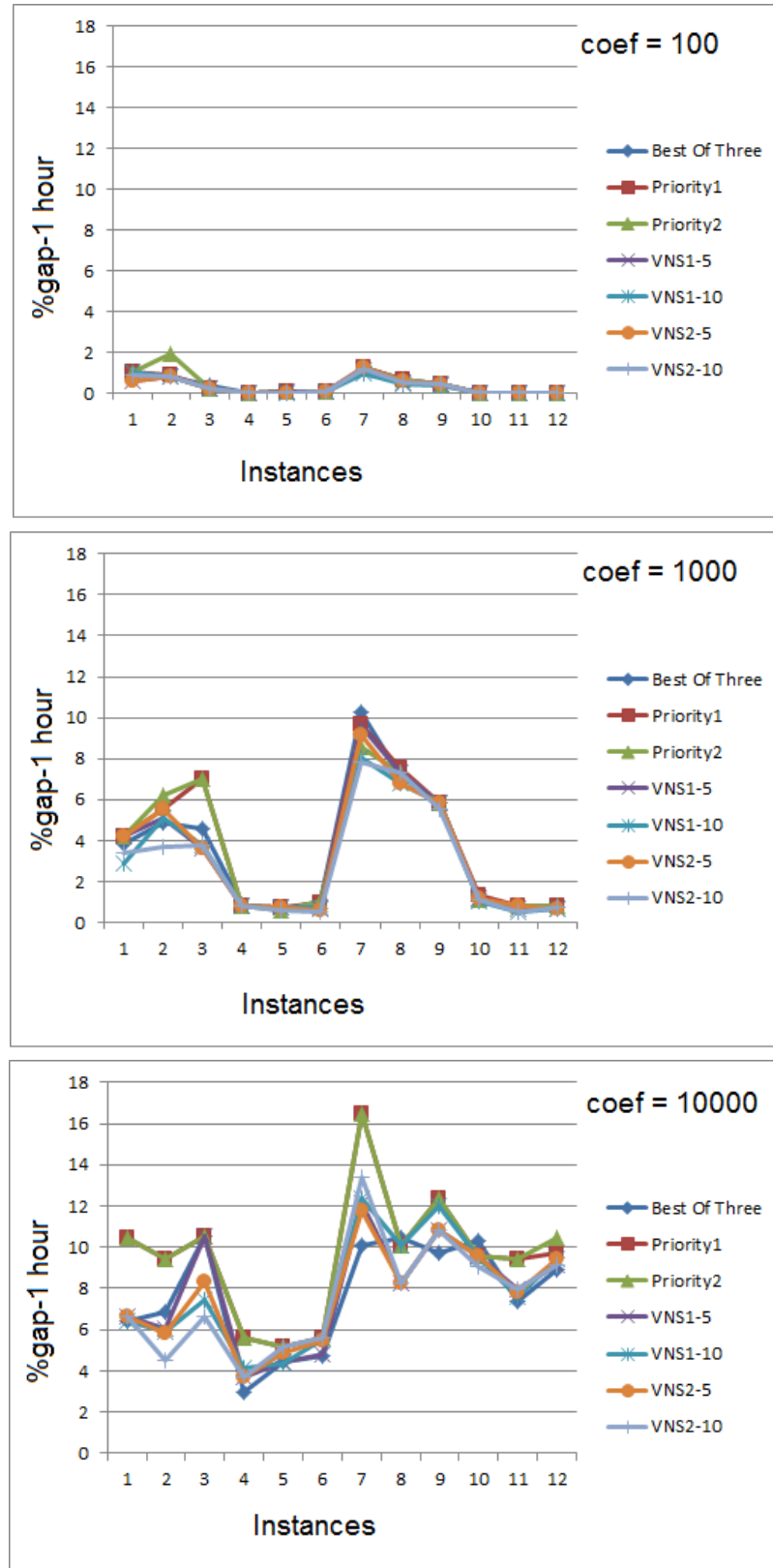


Figure 5.2: Instance-Gap Graphs for 2EDPND Problem

When coefficient is 100 and $d \in [1, 100]$, for both $|N|=30$ and $|N|=40$, all five improvement algorithms find nearly the same $\%gap$. For these instances, our mathematical model solves the instances with the optimality except one instance with node number 40 and $p = 1$, in which model runs out of memory. When $d \in [1, 10]$ for both cases in which node number is 30 and 40, GAMS could not solve the problems with optimality. But its $\%gap$ values are calculated. For one hour bound, in most cases heuristics beat GAMS. However, in ten hours bound, GAMS beats heuristics in most of the instances. In these cases, the difference between the $\%gap$ of the model and heuristics is very small. Additionally, running time of the heuristics are at most 1600 seconds, whereas GAMS runs in ten hours. VNS1 and VNS2 algorithms are the ones that obtain the best solutions.

When coefficient is 1000, GAMS cannot solve the problem in optimality since the time bound is reached. In general, VNS algorithms finds the solutions that are closest to the lower bounds found by the mathematical models. When $d \in [1, 10]$ and $|N|=30$ we beat GAMS in $\%gap$ values. When $|N|=40$, GAMS could not even finish solving the initial LP. So, we cannot calculate its $\%gap$ value. The smallest $\%gap$ found by the improvement algorithms is 3,6 on the average. When $d \in [1, 100]$ and $|N|=30$ heuristics beat GAMS when time limit is one hour. On the other hand, when time limit is ten hours, GAMS is better than the heuristics. When $|N|=40$ heuristics beat GAMS in most cases.

When coefficient is 10000, the problem becomes harder. GAMS cannot find the optimal solutions. When $|N|=30$, GAMS has $\%gap$ for the ten hours time bound only. Except a single instance, heuristics beat GAMS. When $d \in [1, 10]$ VNS2 finds the best solutions for the most cases. Heuristics run at most 75 seconds whereas GAMS run ten hours. When $d \in [1, 100]$ Best Of Three heuristic finds the best solutions for the most cases. Heuristics run at most 50 seconds. When $|N|=40$, the best solutions are generally found by Best Of Three. GAMS cannot find any integer solution in ten hours. Figure 5.3 and Figure 5.4 shows the results of the VNS algorithms. Above graphs are the shows running times for the instances, whereas the below graphs show the gap values calculated for the instances.

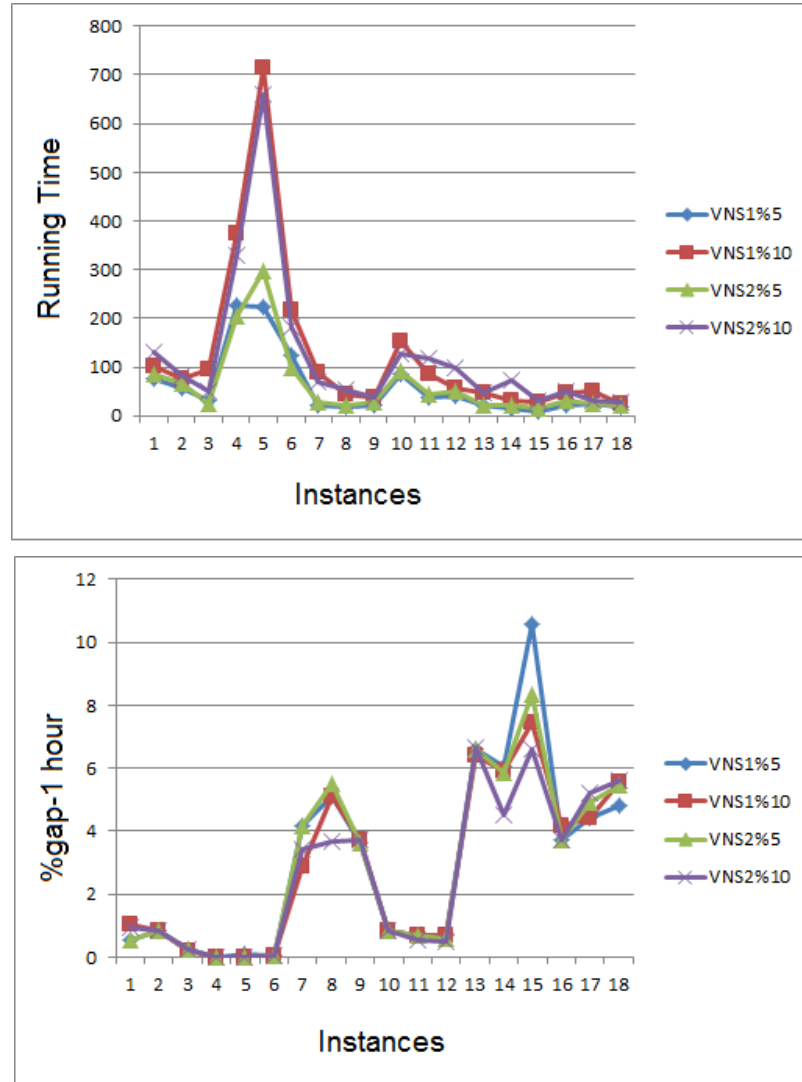


Figure 5.3: Results of VNS algorithms for 2EDPND Problem for instances with $|V| = 30$

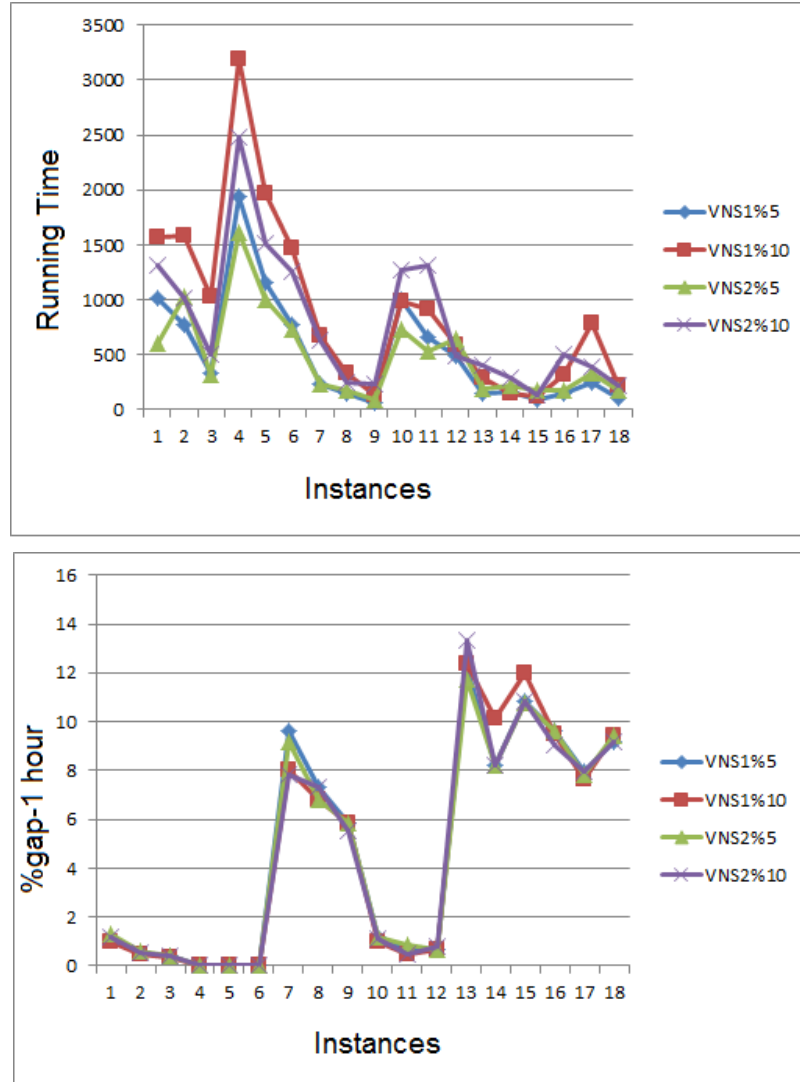


Figure 5.4: Results of VNS algorithms for 2EDPND Problem for instances with $|V| = 40$

Chapter 6

Conclusion

In this thesis, we study two main problems of finding paths between given source and destination pairs. The common trait of these problems is the fixed costs assigned to the each edge of the given network G . Fixed costs are only charged when edge is used for the first time. For the first problem, we find a single path for each commodity using the original costs for the edges. For the second problem, we want to find two edge disjoint paths between each commodity. We denote these paths as; *primary path* and *secondary path*. Original costs of the edges are assigned as the *primary path* cost (c_p) and half of the original costs are assigned as the *secondary path* cost (c_s). We assume that $c_s = 1/2(c_p)$. Commodity set includes all possible node pairs in G as a commodity and a demand value is assigned for each commodity.

In Chapter 2, literature review is illustrated. In the literature we can see many different versions of disjoint paths problems with different objectives. However, we can easily see that none of the problems have considered fixed costs.

IP models are given for the explained problems in Chapter 3. Both models aim to minimize the total cost, which contains the fixed costs of the active edges and routing costs for the paths.

In Chapter 4, heuristic algorithms are described in detail. We use Suurballe's

and Dijkstra's algorithms as a basis of our heuristics. There is one construction algorithm to find an initial feasible solution and eleven improvement algorithms to improve the initial solution. In the improvement algorithms two different approaches are used. One is the local search heuristic and the other is Variable Neighborhood Search Meta-Heuristic. We have three main moves that can be applied to a solution: Adding an edge, deleting an edge and cycling an edge. All three moves are used to search for better solutions. Adding an edge, activates an inactive edge in the solution, deleting an edge, inactivates an edge from the active edge set of the solution. Lastly, cycling an edge inactivates an active edge and activates several inactive edges at the same time. In first six improvement algorithms single moves are used, whereas in last five all three moves are used.

Finally, in Chapter 5 we analyze the computational results of all the heuristics on our test instances. We find out that using more than one move in an improvement algorithm leads to better solutions. When we look at all the results of the heuristics algorithms the algorithms Best Of Three, VNS1 and VNS2 finds the best solutions, whereas Priority2 algorithm has the smallest computational times on the average.

The future work of this thesis can include studying different relations between the costs of primary path and secondary paths. A more realistic but challenging extension might consider capacities added to the edges in order to bound demand transfers. Furthermore, a cost limit can be set for the cost of the secondary path. In order to consider a new and different approach, a tabu search based algorithm can be developed to solve the studied problems.

Bibliography

- [1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993
- [2] R. Bhatia, M. Kodialam and T. V. Lakhsman. Finding disjoint paths with related path costs. *Journal of Combinatorial Optimization*, 12:83-96, 2006
- [3] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: overview and conceptual Comparison. *ACM Computing Surveys*, 35(3): 268-308, 2003
- [4] O. Bräysy. A reactive variable neighborhood search for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 15(4), 347-368, 2003
- [5] F.R.B. Cruz, J. M. Smith. Solving to optimality the uncapacitated fixed-charge network problem, *Computer Operations Research*, 25:67-81,1998
- [6] P. Damcı. Algorithms for the Survivable Telecommunications Network Design Problem Under Dedicated Protection. *Master Thesis*, Bilkent University, 2010
- [7] C. Duhamel. Solving the uncapacitated fixed-charge network flow with metaheuristics. *4th Metaheuristics International Conference*, 685-689, 2001
- [8] 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, 2009
- [9] P. Hansen and N. Mladenovic. A tutorial on variable neighborhood search. *Les Chaîers du GERAD* ,G(46), 2003

- [10] V. C. Hemmelmayr, K. F. Doerner and R. F. Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operations Research*, 195(3): 791-802, 2009
- [11] K. Holmberg and J. Hellstrand. Solving the uncapacitated network design problem by a Lagrangian heuristic and branch-and-bound. *Operations Research* 46(2): 247-259, 1998
- [12] S. W. Lee and C. S. Wu. A K-best paths algorithm for highly reliable communication networks. *IEICE Transactions on Communications*, E82-B(4):586-589, 1999
- [13] C. H. Li, S. Thomas and D. Simchi-Levi. Finding disjoint paths with different path-costs: complexity and algorithms. *Networks*, 22:653-667, 1992
- [14] C. H. Li, S. Thomas and D. Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete App. Math*, 26(1): 105-115, 1990
- [15] İ. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513-623, 1996
- [16] Y. Perl and Y. Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the Association for Computing Machinery*, 25(1):1-9, 1978
- [17] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125-145, 1974
- [18] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325-336, 1984
- [19] D. Xu, Y. Chen, Y. Xiong, C. Qiao, X. He. On the complexity of and algorithms for finding the shortest path with a disjoint counterpath. *IEEE/ACM Transactions on Networking*, 14(1), 2006
- [20] S. Q. Zheng, B. Yang, M. Yang and J. Wang. Finding minimum-cost paths with minimum sharability. *IEEE INFOCOM*, 1532-1540, 2007

Appendix A

Computational Results

Tables A.1 to A.12 show the results of all proposed heuristics for the problem of SPND. Tables A.13 to A.24 show the results of all proposed heuristics for the problem of 2EDPND. All tables are structured identically. Each table shows the results of three instances. The first column presents the density level of the instance. The second column presents the number of edges in the instance. The remaining columns represent the results of the heuristics. Each column presents number of active edges, running time in seconds, %gap-1 calculated for the 1 hour limit and %gap-10 calculated for the 10 hour limit. Columns of the heuristics Best Of Three, Priority 1 and Priority 2 additionally present the number of edges added (Add), number of edges deleted (Delete) and number edges cycled (Cycle). Heuristics VNS1 and VNS2 additionally present the exact time of the best solution found (Best). For the running time of the mathematical models RLE stands for resource limit exceeded and OOM stands for out of memory. In these cases, %gap values are calculated according to the lower bounds obtained from the models.

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOfThree	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10
1	435	30 0,07 36,9 -	48 1,05 19,06 -	41 7,49 6,29 -	30 0,12 36,9 -	30 0,12 36,9 -	32 0,36 9,47 -	30 1,79 9,73 -	35 13,83 0,27 -	34 9,5 0,47 -	34 8,75 0,47 -	34 14 0,36 -	34 18,45 0,47 -	34 13,86 0,47 -	34 19,49 0,47 -	34 2042 0 -
0,75	322	32 0,12 15,82 15,32	48 0,97 8,42 7,96	42 6,14 2,53 2,1	32 0,3 15,82 15,32	32 0,21 15,82 15,32	33 0,54 6,99 6,53	33 1,38 6,99 6,53	38 13,27 0,52 0,09	38 8,68 0,52 0,09	38 6,95 0,52 0,09	38 14,55 0,52 0,09	38 18,94 0,52 0,09	38 10,83 0,52 0,09	37 15,14 0,46 0,03 9,66	37 15484,5 0,57 0
0,5	192	38 0,08 3,3 -	42 0,56 1,56 -	43 1,83 1,28 -	38 0,26 3,3 -	38 0,24 3,3 -	38 0,43 2,31 -	37 1,28 1,73 -	39 5,72 0 -	39 3,95 0 -	39 3,76 0 -	39 5,91 0 -	39 10,46 0 -	39 6,29 0 -	39 8,81 0 -	39 151,57 0 -

Table A.1: Computational Results for SPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,10\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	VNS2 Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best
1	435	56 0,09 7,23 —	92 0,97 2,97 —	81 13,1 1,39 —	52 0,32 7,1 —	52 0,76 7,1 —	55 0,65 3,36 —	55 2,88 3,29 —	74 30,24 0,12 — 24 7 9	73 19,97 0,02 — 21 1 25	73 16,35 0,02 — 21 0 26	73 34,35 0,02 — — —	73 50,73 0,02 — — —	73 31,67 0,02 — — —	73 48,63 0,02 — — —	73 31,59 0 — —
0,75	322	60 0,06 2,74 —	78 0,77 1,34 —	74 6,1 0,76 —	56 0,33 2,64 —	56 0,83 2,64 —	58 0,5 1,8 —	59 2,05 1,79 —	71 14,04 0,03 — 13 4 3	70 11,68 0,07 — 13 0 12	70 9,37 0,07 — 13 0 12	70 26,24 0,04 — 16,19 —	69 40,44 0,07 — 20,46 —	70 24,26 0,04 — 16,27 —	71 54,09 0,03 — 34,98 —	71 102,3 0 — —
0,5	215	82 0,12 0,94 —	86 0,58 0,82 —	85 1,21 0,67 —	71 0,48 0,32 —	72 1,62 0,32 —	71 0,67 0,3 —	72 2,66 0,3 —	75 7,69 0,01 — 3 10 0	75 5,57 0,04 — 3 0 11	75 4,3 0,04 — 3 0 11	75 15,48 0,01 — 8,98 —	75 26,14 0,03 — 8,48 —	75 15,44 0,04 — — —	75 22,18 0,04 — — —	74 44,71 0 — —

Table A.2: Computational Results for SPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,100\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	435	29 0,1 38,65 36,15	32 1,59 35,91 33,46	31 2,39 35,91 33,46	29 0,12 38,65 36,15	29 0,12 38,65 36,15	29 0,43 11,72 9,7	29 1,83 11,72 9,7	29 10,5 9,9 7,92 1 1 7	29 5,35 9,9 7,92 1 0 10	29 3,87 9,9 7,92 1 0 10	29 8,8 9,9 7,92 —	29 8,63 9,9 7,92 —	29 7,51 9,9 7,92 —	29 11,48 9,9 7,92 —	RLE — 88,82 14,21
0,75	323	29 0,07 24,8 16,98	30 1,11 23,1 15,39	30 1,64 22,88 15,18	29 0,13 24,8 16,98	29 0,08 24,8 16,98	29 0,48 14,26 7,1	29 1,3 14,16 7,01	29 5 14,16 7,01 0 0 4	29 2,18 14,16 7,01 0 0 4	29 2,11 14,16 7,01 0 0 0	29 5,04 14,16 7,01 —	29 6,66 14,16 7,01 —	29 5,62 14,16 7,01 —	29 7,81 14,16 7,01 —	RLE — — —
0,5	217	29 0,07 12,77 12,33	30 0,64 10,82 10,39	30 1,12 10,82 10,39	29 0,13 12,77 12,33	29 0,14 12,77 12,33	29 0,39 5,33 4,92	29 1,37 4,01 3,61	29 3,71 4,01 3,61 0 0 4	29 2,07 4,01 3,61 0 0 4	29 1,74 4,01 3,61 0 0 4	29 3,92 4,01 3,61 —	29 5,28 4,01 3,61 —	29 4,93 4,01 3,61 —	29 6,59 4,01 3,61 —	RLE — 15,29 11,46

Table A.3: Computational Results for SPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,10\}$

p	m	One Step	Add	Add Best	Delete	Delete Best	Cycle	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10
1	435	30 0,07 30,35 —	50 1,24 18,83 —	44 10,61 7,35 —	30 0,11 30,35 —	30 0,4 17,86 —	31 1,93 20,63 —	37 19,17 0,4 —	36 11,31 0,95 —	36 10,22 0,95 —	36 10,22 0,95 —	36 19,3 0,95 —	36 27,02 0,95 —	36 18,05 0,21 —	36 22,36 0,95 —	37 585,65 0 —
0,75	325	29 0,06 18,05 17,91	43 0,95 6,99 6,86	37 5,22 2,26 2,14	29 0,13 18,05 17,91	30 0,38 10,74 10,6	30 1,46 10,74 10,6	37 11,83 0,17 0,4	38 9,49 0,36 0,24	38 7,65 0,36 0,24	38 7,65 0,36 0,24	38 11,81 0,36 0,24	38 20,96 0,36 0,24	38 12,28 0,36 0,24	38 16,04 0,36 0,24	36 5720 0,14 0
0,5	230	32 0,09 3,86 —	37 0,75 1,83 —	36 2,15 0,91 —	32 0,26 3,86 —	32 0,43 2,15 —	32 0,98 2,15 —	35 4,22 0 —	35 4,57 0 —	35 3,71 0 —	35 3,71 0 —	35 7,3 0 —	35 10,04 0 —	35 7,06 0 —	35 11,12 0 —	34 19,28 0 —

Table A.4: Computational Results for SPND Problem with Node Number = 30 , Coefficient= 1000. Demand $\in \{1,100\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10
1	435	29 0,06 72,36 71,89	29 1,11 72,36 71,89	29 1,12 72,36 71,89	29 0,13 72,36 71,89	29 0,08 72,36 71,89	29 0,45 30,9 30,54	29 1,76 30,9 30,54	29 9,92 30,9 30,54	29 2,58 30,9 30,54	29 2,58 30,9 30,54	29 6,64 30,9 30,54	29 8,15 30,9 30,54	29 6,36 30,9 30,54	29 9,54 30,9 30,54	RLE — 97,09 72,13
0,75	334	29 0,06 40,86 40,82	29 0,87 40,86 40,82	29 0,9 40,86 40,82	29 0,08 40,86 40,82	29 0,09 40,86 40,82	29 0,4 27,86 27,82	29 1,49 27,86 27,82	29 7,14 27,64 27,6	29 2,38 27,86 27,82	29 2,11 27,86 27,82	29 6,54 27,64 27,6	29 7,76 27,64 27,6	29 6,93 27,64 27,6	29 9,11 27,64 27,6	RLE — — —
0,5	209	29 0,06 39,51 39,22	29 0,61 39,51 39,22	29 0,59 39,51 39,22	29 0,14 39,51 39,22	29 0,13 39,51 39,22	29 0,42 31,09 30,82	29 1,15 31,09 30,82	29 4,06 31,09 30,82	29 2,05 31,09 30,82	29 1,91 31,09 30,82	29 4,28 31,09 30,82	29 5,18 31,09 30,82	29 4,66 31,09 30,82	29 7,81 30,68 30,4	RLE — — 43,19

Table A.5: Computational Results for SPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,10\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	435	29 0,06 46,64 46,42	30 1,1 46,1 45,89	30 1,96 45,01 44,8	29 0,13 46,64 46,42	29 0,14 46,64 46,42	29 0,42 14,95 14,79	29 1,81 11,8 11,63	30 11,95 9,83 9,67	30 4,28 9,83 9,67	30 3,94 9,83 9,67	30 9,63 9,83 9,67	30 9,71 9,83 9,67	30 9,09 9,83 9,67	30 12,11 9,83 9,67	RLE – – –
0,75	326	29 0,07 26,32 26,3	31 0,9 23,82 23,8	30 1,55 19,12 19,11	29 0,09 26,32 26,3	29 0,08 26,32 26,3	30 0,42 12,23 12,22	30 1,67 13,09 13,08	29 9,16 11,75 11,74	29 2,44 12,39 12,37	29 3,8 12,39 12,37	29 6,8 10,62 10,6	29 7,32 10,62 10,6	29 6,77 10,62 10,6	29 8,81 10,62 10,6	RLE – 84,9 78,52
0,5	236	29 0,11 9,5 8,74	29 0,73 9,5 8,74	29 0,76 9,5 8,74	29 0,15 9,5 8,74	29 0,16 9,5 8,74	29 0,4 7,35 6,6	29 1,28 7,35 6,6	29 4,38 7,23 6,49	29 1,98 7,35 6,6	29 2,01 7,35 6,6	29 5,12 7,23 6,49	29 6,34 7,23 6,49	29 6,49 7,23 6,49	29 8,89 6,99 6,25	RLE – – 9,6

Table A.6: Computational Results for SPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,100\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	780	40 0,1 31,5 31,39	77 3,87 14,66 14,56	65 70,58 5,39 5,3	40 0,27 31,5 31,39	40 0,34 31,5 31,39	45 1,09 10,52 10,43	43 5,39 10,72 10,63	57 110,52 0,82 0,73	57 60,89 0,91 0,83	57 57,99 0,91 0,83	57 94,93 0,91 0,83	57 139,72 0,91 0,83	57 95,55 0,91 0,83	57 132,35 0,91 0,83	RLE – – 0,84
0,75	599	44 0,09 13,52 12,99	66 2,99 6,28 5,78	61 38,33 3,13 2,65	43 0,36 13,33 12,8	43 0,5 13,33 12,8	46 1,03 7,28 6,78	47 4,8 7,28 6,78	50 74,69 1,01 0,54	50 40,6 0,98 0,51	51 34,49 1,18 0,71	49 62,79 0,98 0,51	50 84,72 0,9 0,43	50 60,32 0,98 0,51	51 105 0,9 0,43	RLE – 0,95 0,17
0,5	410	49 0,12 7,08 6,59	64 2,56 4,77 4,29	63 21,45 2,18 1,71	46 0,53 6,72 6,23	46 1,11 6,72 6,23	48 0,98 4,12 3,64	48 3,71 4,03 3,55	56 36,02 0,87 0,41	56 28,06 0,83 0,37	56 26,69 0,87 0,41	56 49,05 0,83 0,37	56 71,65 0,83 0,37	56 38,59 0,83 0,37	57 97,31 0,77 0,31	RLE – 0,66 0,08

Table A.7: Computational Results for SPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,10\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	780	39 0,11 48,97 44,96	48 3,77 44,16 40,28	44 18,19 32,23 28,68	39 0,21 48,97 44,96	39 0,14 48,97 44,96	39 1,07 33,39 29,81	39 4,18 26,14 22,75	41 40,98 21,06 17,81	41 22,72 13,27 10,23	41 18,82 13,27 10,23	41 38,44 13,27 10,23	42 107,59 10,56 7,59 67,44	41 51,24 13,12 10,08 26,77	42 80,9 10,79 7,81 32,76	RLE - - -
0,75	595	39 0,12 38,76 35,11	45 3,06 33,38 29,88	42 9,9 19,35 16,21	39 0,15 38,76 35,11	39 0,14 38,76 35,11	40 0,79 14,77 11,76	40 3,33 14,35 11,34	40 52,62 8,82 5,96	40 14,57 8,82 5,96	40 17,89 8,84 5,98	40 21,09 8,82 5,96	40 33,9 8,82 5,96	40 25,23 8,82 5,96	40 43,63 8,82 5,96	RLE - - -
0,5	377	39 0,11 23,37 19,49	45 1,96 18,42 14,69	43 7,61 11,64 8,12	39 0,26 23,37 19,49	39 0,21 23,37 19,49	40 0,77 16,23 12,57	40 2,37 19,01 15,27	41 25,57 7,6 4,21	41 7,95 10,81 7,32	41 6,96 10,81 7,32	40 52,23 10,43 6,96	40 52,23 10,43 6,96	41 20,99 10,81 7,32	41 52,11 8,03 4,62 28,03	RLE - - -

Table A.9: Computational Results for SPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,10\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10 Add Delete Cycle	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	780	41 0,1 34,36 34,19	73 3,63 13,55 13,4	61 59,29 4,3 4,17	41 0,3 34,36 34,19	41 0,35 34,36 34,19	41 1,04 14,15 14	43 4,41 18,03 17,89	53 102 1,12 1 11 1 17	52 59,15 0,65 0,52 12 0 28	52 47,44 0,65 0,52 12 0 28	52 91,35 0,65 0,52 —	52 143,39 0,65 0,52 —	52 85,18 0,65 0,52 —	52 158,21 0,65 0,52 —	RLE — 0,65 0,36
0,75	596	43 0,1 21,06 20,56	71 2,89 8,62 8,17	61 40,87 3,59 3,16	42 0,39 21,02 20,51	42 0,56 21,02 20,51	44 1,05 9,86 9,4	44 4,45 9,39 8,94	52 68,67 1,65 1,23 13 6 6	51 45,32 1,21 0,78 12 1 26	51 38,14 1,21 0,78 12 0 27	51 64,78 1,11 0,69 52,09	51 89,44 1,11 0,69 47,7	51 72,28 1,11 0,69 54,2	51 93,41 1,11 0,69 51,17	RLE — 2,09 0,23
0,5	406	53 0,1 8,97 8,9	68 1,94 5,5 5,44	61 13,01 3,68 3,62	51 0,47 8,38 8,31	51 0,95 8,38 8,31	50 0,74 4,97 4,91	51 3,9 5,05 4,99	54 39,68 1,2 1,14 7 6 7	54 21,09 1,2 1,14 6 1 17	54 21,35 1,2 1,14 6 2 16	54 40,9 1,2 1,14 —	54 70,34 1,2 1,14 —	54 35,28 1,2 1,14 —	54 61,98 1,2 1,14 —	RLE — 2,73 1,38

Table A.10: Computational Results for SPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,100\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10
1	780	39 0,11 93,07 79,01	39 3,77 93,07 79,01	39 4,03 93,07 79,01	39 0,21 93,07 79,01	39 0,17 93,07 79,01	39 1,09 50,37 39,43	39 4,61 51,54 40,51	39 73,52 49,92 39,01	39 7,49 51,54 40,51	39 7,25 51,54 40,51	39 21,73 49,92 39,01	39 38,14 49,92 39,01	39 25,89 49,92 39,01	39 48,36 49,92 39,01	RLE – – 94,46
0,75	591	39 0,1 57,84 41,88	39 2,87 57,84 41,88	39 2,96 57,84 41,88	39 0,18 57,84 41,88	39 0,13 57,84 41,88	39 0,94 43,1 28,64	39 3,06 43,01 28,56	39 24,69 43,01 28,56	39 5,63 43,01 28,56	39 5,37 43,01 28,56	39 13 43,01 28,56	39 23,69 43,01 28,56	39 16,03 43,01 28,56	39 35,18 43,01 28,56	RLE – – –
0,5	378	39 0,11 48,98 24,62	39 1,93 48,98 24,62	39 2,02 48,98 24,62	39 0,21 48,98 24,62	39 0,22 48,98 24,62	39 0,72 45,54 21,74	39 2,24 44,44 20,82	39 10,86 44,44 20,82	39 4,08 44,44 20,82	39 3,63 44,44 20,82	39 9,87 44,44 20,82	39 16,82 44,44 20,82	39 10,64 44,44 20,82	39 24,34 44,44 20,82	RLE – – 25,36

Table A.11: Computational Results for SPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,10\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	780	39 0,1 57,64 40,67	45 3,63 48,48 32,5	44 17,63 36,36 21,9	39 0,14 57,64 40,67	39 0,14 30,84 40,67	40 0,83 31,08 16,76	40 4,74 17,5 16,98	42 82,74 16,18 4,86	41 12,47 16,18 3,68	41 11,58 14,57 3,68	41 45,83 16,18 2,24	41 53,65 16,18 3,68	41 38,28 14,57 3,68	41 123,43 14,57 3,68	RLE - - -
0,75	590	39 0,1 27,76 18,84	43 2,95 22,85 14,27	43 11,89 18,49 10,22	39 0,15 27,76 18,84	39 0,13 27,76 18,84	40 0,9 17,83 9,6	39 2,48 18,25 9,99	40 42,38 11,71 3,91	40 14,25 11,43 3,64	40 15,53 11,43 3,64	40 19,7 11,43 3,64	40 66,23 11,06 3,3	40 23,94 11,43 3,64	40 43,61 11,43 3,64	RLE - - -
0,5	417	39 0,11 20,25 19,32	47 2,03 16,31 15,41	44 9,95 7,55 6,72	39 0,16 20,25 19,32	39 0,14 20,25 6	40 0,95 6,82 4,9	40 3,36 5,72 0,46	40 30,31 1,24 0,48	40 14,11 1,26 0,48	40 12,89 1,26 0,48	40 18,88 1,26 0,48	40 25,27 1,26 0,48	40 18,11 1,26 0,48	40 29,46 1,26 0,48	RLE - - 43,19

Table A.12: Computational Results for SPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,100\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Best
1	435	50 0,13 7,61 7,55	63 2,79 4,94 4,88	60 23,79 2,64 2,58	50 0,41 7,61 7,55	50 0,42 7,61 7,55	54 1,12 3,35 3,3	53 4,15 4,23 4,17	56 56,65 1,07 6	56 21,68 1,05 0,99	56 18,25 1,05 0,99	57 76,41 0,57 0,51 47,53	56 101,59 1,05 0,99 —	57 86,16 0,57 0,51 53,99	56 129,93 0,93 0,87 52,45	RLE — 1,22 0,52
0,75	322	58 0,14 1,93 1,8	62 2 1,82 1,7	61 6,33 1,74 1,62	56 0,53 1,78 1,65	56 1,06 1,78 1,65	57 1,12 1,1 0,98	57 3,65 1,06 0,94	57 23,55 0,85 0,72 1 2 4	57 10,86 0,92 0,8 1 1 5	57 8,85 1,93 1,8 1 0 6	57 56,74 0,85 0,72 25,6	57 77,52 0,85 0,72 14,13	57 65,3 0,85 0,72 32,57	57 81,66 0,85 0,72 30,04	RLE — 1,51 0,51
0,5	192	64 0,14 2,76 2,64	66 0,97 2,74 2,63	65 1,75 2,57 2,45	51 0,58 1,56 1,44	53 3,64 0,79 0,67	55 1,14 0,99 0,88	56 6,03 0,81 0,7	54 30,68 0,35 0,24 2 12 1	55 14,8 0,23 0,11 2 3 13	54 16,75 0,2 0,09 2 4 12	55 32,36 0,23 0,11 —	54 94,16 0,2 0,09 48,25	55 26,42 0,23 0,11 —	55 50,54 0,23 0,11 —	RLE — 0,06 0,02

Table A.13: Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,10\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best	edge # time gap-1 gap-10 Best
1	435	103 0,19 1,11 —	117 2,51 0,73 —	115 28,31 0,51 —	99 1 0,95 —	98 4,29 0,95 —	100 1,62 0,53 —	101 12,29 0,56 —	105 99,87 0,02 —	105 66,84 0,02 —	105 66,98 0,02 —	105 226,93 0,02 —	105 372,95 0,02 —	105 204,96 0,02 —	105 329 0,02 —	105 322 0 —
0,75	322	111 0,13 1,13 —	116 1,81 1,03 —	116 8,93 0,95 —	91 1 0,43 —	92 13,43 0,42 —	96 1,69 0,3 —	98 17,12 0,27 —	100 90 0,09 —	100 46,89 0,09 —	100 41,81 0,09 —	100 223,3 0,09 —	100 714,32 0,02 —	100 295,89 0,02 —	100 661,45 0,02 —	100 45 0 —
0,5	215	117 0,14 0,51 —	118 0,89 0,5 —	118 1,56 0,5 —	104 1,02 0,09 —	105 8,99 0,06 —	105 1,57 0,09 —	106 11,73 0,06 —	106 37,06 0,05 —	106 19,33 0,05 —	106 17,66 0,05 —	106 123,74 0,05 —	106 217,38 0,05 —	106 98,76 0,05 —	106 181,62 0,05 —	104 55,59 0 —

Table A.14: Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 100, Demand $\in \{1,100\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	435	41 0,17 22,71 22,67	46 2,81 19,45 19,42	45 11,64 19,2 19,16	39 0,33 21,56 21,52	39 0,5 21,56 21,52	40 0,97 5,87 5,83	39 4,68 7,29 7,25	40 74,94 3,84 3,8 0 3 16	40 13,56 4,18 4,15 1 0 17	40 11,19 4,18 4,15 1 0 17	39 87,84 2,86 2,82 65,14	40 23,24 4,18 4,15 —	40 26,85 4,18 4,15 —	39 71,37 3,43 3,4 36,72	RLE — — 12
0,75	323	45 0,16 18,96 18,72	46 1,98 18,34 18,1	46 3,51 18,27 18,04	38 0,28 14,21 13,98	39 0,97 12,78 12,56	43 0,96 11,77 11,55	43 3,69 11,77 11,55	39 43,14 4,87 4,66 2 8 6	39 11,74 5,51 5,3 1 5 9	38 18,82 6,2 5,99 1 4 10	39 43,7 5,12 4,91 23,53	39 19,23 5,12 4,91 13,67	39 21,4 5,51 5,3 —	37 54,52 3,7 3,49 28,73	RLE — — 10,46
0,5	217	44 0,14 11,83 11,81	45 1,23 11,48 11,46	45 2,31 11,48 11,46	43 0,17 11,66 11,64	43 0,39 9,58 9,56	43 0,94 8,23 8,22	42 3 7,66 7,64	42 21,72 4,59 4,57 1 3 6	43 7,22 7,01 6,99 1 0 7	43 6,76 7,01 6,99 1 1 7	42 36,45 3,74 3,72 12,86	42 22,11 3,63 3,62 13,59	41 27,58 3,65 3,64 17,27	42 39,31 3,74 3,72 14,69	RLE — — 23,48

Table A.15: Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,10\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10
1	435	53 0,13 9,98 9,83	68 2,65 5,36 5,22	62 22,17 3,56 3,43	49 0,61 9,62 9,48	49 1,5 9,54 9,39	50 6,16 4,93 4,79	50 1,29 4,6 4,46	56 89,73 0,87 0,74	58 50,33 0,84 0,7	58 48,49 0,84 0,7	58 85,56 0,84 0,7	58 153,37 0,84 0,7	58 92,14 0,84 0,7	58 127,87 0,84 0,7	RLE – 2,01 0,28
0,75	325	51 0,19 4,52 4,43	60 2,14 2,47 2,38	59 14,45 1,51 1,43	50 0,59 4,41 4,33	50 0,86 4,41 4,33	51 4,29 2,98 2,9	52 1,15 3,19 3,1	52 53,76 0,57 0,48	53 26,48 0,72 0,64	53 32,56 0,61 0,53	53 38,72 0,72 0,64	53 84,78 0,72 0,64	53 43,22 0,72 0,64	52 117,05 0,57 0,48	RLE – 5,19 0,31
0,5	230	57 0,14 4,21 3,92	62 1,4 3,22 2,94	61 5,49 2,35 2,07	55 0,5 3,45 3,17	54 1,45 3,32 3,04	55 3,59 2,72 2,44	56 1,03 2,79 2,51	58 19,64 1,07 0,79	55 15,8 0,95 0,67	58 14,57 1,02 0,75	55 40,61 0,7 0,43	56 56,2 0,72 0,45	56 50,71 0,59 0,31	55 98,57 0,51 0,23	RLE – 3,57 0,09

Table A.16: Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 1000, Demand $\in \{1,100\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle	edge # time gap-1 gap-10 Add Delete Cycle
1	435	41 0,15 41,58 41,06	41 2,89 41,58 41,06	41 2,81 41,58 41,06	34 0,23 17,68 17,25	34 0,98 17,68 17,25	35 1,07 13,45 13,03	35 4,28 11,83 11,42	33 48,42 6,41 6,02 0 8 4	34 7,03 10,47 10,07 0 1 10	34 10,95 10,47 10,07 0 1 10	33 20,92 6,63 6,24 13,22	33 47,77 6,42 6,03 24,55	33 20,44 6,63 6,24 12,46	33 47,34 6,66 6,27 13,36	RLE – – 5,85
0,75	334	41 0,13 36,2 36,2	41 2 36,2 36,2	41 2,09 36,2 36,2	38 0,2 23,39 23,39	36 0,55 12,86 12,85	36 1,03 8,19 8,18	35 4,35 9,43 9,42	34 33,04 6,84 6,84 0 7 5	35 6,3 9,43 9,42 0 0 10	35 5,91 9,43 9,42 0 0 10	33 16,57 6,08 6,07 10,38	33 30,55 5,89 5,88 11,77	33 21,48 5,86 5,86 12,13	33 72,07 4,53 4,53 45,86	RLE – – 32,8
0,5	209	42 0,12 27,23 26,83	42 1,19 27,23 26,83	42 1,17 27,23 26,83	36 0,23 14,84 14,47	37 0,53 13,96 13,6	36 0,99 9,75 9,4	36 3,51 10,56 10,21	36 17,18 10,56 10,21 0 6 2	36 4,83 10,56 10,21 0 0 8	36 4,72 10,56 10,21 0 0 8	36 9,98 10,56 10,21 –	34 27,13 7,47 7,13 15,72	36 15,1 8,32 7,97 9,23	35 30,77 6,63 6,29 12,48	RLE – – 32,62

Table A.17: Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,10\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority		VNS1 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best	
										1	2						
1	435	41	45	43	37	37	38	38	37	38	38	37	37	37	37	RLE	
		0,15	2,63	6,97	0,23	1,15	3,99	1,15	3,99	53,2	6,92	6,47	20,42	46,88	32,25	51,66	-
		16,64	14,3	11,65	12,53	5,85	5,63	5,85	5,63	3	5,63	5,63	3,73	4,16	3,73	3,73	-
		16,39	14,06	11,42	12,29	5,63	5,41	5,63	5,41	2,78	5,41	5,41	3,51	3,94	3,51	3,51	23,2
0,75	326	42	43	43	39	39	39	39	39	40	40	39	39	39	40	RLE	
		0,16	1,98	3,76	0,23	0,95	3,73	0,95	3,73	24,27	8,85	7,98	25,73	49,33	26,37	32,07	-
		10,92	10,26	10,92	7,95	5,46	5,35	5,46	5,35	4,44	5,2	5,2	4,44	4,4	54,91	5,2	-
		10,92	10,26	10,92	7,95	5,46	5,35	5,46	5,35	4,44	5,2	5,2	4,44	4,4	54,91	5,2	-
0,5	236	44	44	44	40	40	40	40	40	40	40	41	40	40	40	RLE	
		0,14	1,4	1,4	0,28	0,99	4,1	0,99	4,1	27,34	5,65	5,46	19,14	25,66	21,38	27,87	-
		12,6	12,6	12,6	7,35	7,2	5,6	7,2	5,6	4,74	5,6	5,6	4,8	5,57	5,44	5,6	-
		11,96	11,96	11,96	6,74	6,59	5	6,59	5	4,14	5	5	4,21	4,97	4,84	5	5,83
0,5	236	44	44	44	40	40	40	40	40	40	40	41	40	40	40	RLE	
		0,14	1,4	1,4	0,28	0,99	4,1	0,99	4,1	27,34	5,65	5,46	19,14	25,66	21,38	27,87	-
		12,6	12,6	12,6	7,35	7,2	5,6	7,2	5,6	4,74	5,6	5,6	4,8	5,57	5,44	5,6	-
		11,96	11,96	11,96	6,74	6,59	5	6,59	5	4,14	5	5	4,21	4,97	4,84	5	5,83

Table A.18: Computational Results for 2EDPND Problem with Node Number = 30 , Coefficient= 10000, Demand $\in \{1,100\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Best
1	780	68 0,23 5,75 —	92 11,28 3,59 —	84 202,56 1,63 —	68 1,12 5,75 —	68 1,08 5,75 —	68 2,5 4,52 —	69 8,81 4,74 —	83 319,46 1,3 —	83 14 2 18	84 295,42 1,23 —	83 1008,89 1,1 —	85 1565,72 0,98 —	83 608,45 1,3 —	82 1312,67 1,17 —	OOM 300 — —
0,75	599	79 0,28 4,7 4,69	93 8,21 2,69 2,68	89 98,86 2,14 2,13	75 1,48 4,1 4,09	75 5,53 4,07 4,07	79 3,58 2,19 2,18	77 15,3 2,67 2,66	84 234,67 0,67 0,66	84 189,58 0,65 0,64	84 152,48 0,65 0,64	83 775,84 0,55 0,54	84 1590,31 0,48 0,47	85 1028,25 0,58 0,57	83 1022,86 0,51 0,51	RLE — 1,09 0,59
0,5	410	78 0,28 2,5 2,43	87 5,65 1,61 1,54	86 46,49 1,41 1,34	73 1,28 1,77 1,7	78 6,02 2,5 2,43	75 2,64 1,34 1,27	75 13,35 1,34 1,27	81 165,73 0,42 0,36	82 128,68 0,42 0,35	82 105,42 0,42 0,35	82 330,79 0,42 0,35	81 1026,48 0,36 0,3	82 322,12 0,42 0,35	82 500,11 0,42 0,35	RLE — 0,79 0,1

Table A.19: Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,10\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority1	Priority2	VNS1	VNS1	VNS2	VNS2	VNS2	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10
1	780	154 0,2 0,79 —	179 11,68 0,37 —	178 278,54 0,32 —	148 3,23 0,69 —	148 19,29 0,67 —	150 4,44 0,52 —	149 45,12 0,5 —	165 862,97 0,02 —	165 564,09 0,02 —	165 488,12 0,02 —	165 3191,56 0,02 —	165 1613,64 0,02 —	165 2479,56 0,02 —	165 2479,56 0,02 —	165 2479,56 0,02 —	OOM 2494 — —
0,75	568	179 0,22 0,82 —	185 7,32 0,67 —	185 50,48 0,65 —	153 3,62 0,38 —	156 73,78 0,36 —	157 5,32 0,21 —	159 120,12 0,19 —	162 788,57 0,02 —	160 341,51 0,02 —	160 284,4 0,03 —	160 1963,26 0,02 —	160 999,02 0,02 —	160 1514,45 0,02 —	160 1514,45 0,02 —	160 1514,45 0,02 —	162 228 0 —
0,5	379	178 0,22 0,84 —	179 4,45 0,84 —	179 8,21 0,84 —	149 3,79 0,12 —	151 80,85 0,7 —	155 513 0,11 —	155 102,3 0,08 —	157 466,09 0,01 —	157 201,18 0,01 —	157 195,32 0,01 —	157 1473,16 0,01 —	157 728,43 0,01 —	157 1261,61 0,01 —	157 1261,61 0,01 —	157 1261,61 0,01 —	158 780 0 —

Table A.20: Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 100, Demand $\in \{1,100\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10 Add Delete Cycle	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0,1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	780	56 0,25 25,98 20,89	65 11,44 22,86 17,9	61 65,24 16,2 11,51	55 0,42 25,84 20,77	55 0,67 25,84 20,77	60 2,87 15,02 10,38	60 19,87 14,31 9,7	55 413,08 10,25 5,8 5 7 10	54 67,79 9,69 5,26 1 6 23	56 235,8 8,58 4,2 5 6 26	54 240,09 9,62 5,2 127,25	53 676,18 8,03 3,67 399,97	55 228,58 9,18 4,78 116,86	54 631,24 7,84 3,49 330,57	RLE - - -
0,75	595	55 0,27 15,19 11,61	60 8,47 14,24 10,69	61 52,51 11,88 8,4	53 0,44 14,86 11,29	53 0,97 14,86 11,29	57 2,73 9,76 6,34	55 15,38 9,05 5,66	56 249,15 7,18 3,84 4 4 10	54 45,17 7,57 4,22 1 2 15	54 82,55 7,3 3,96 2 2 17	54 153,06 7,3 3,96 70,05	55 329,36 6,83 3,51 123,02	55 181,93 6,83 3,51 92,27	54 250,15 7,3 3,96 60,14	RLE - - -
0,5	377	59 0,23 10,38 8,4	60 5,18 9,77 7,79	60 10,36 9,77 7,79	55 0,5 7,85 5,91	55 1,73 8,14 6,19	56 2,3 5,83 3,92	55 11,25 6,17 4,26	56 112,84 5,83 3,91 2 5 6	56 28,17 5,82 3,91 1 0 12	56 26,93 5,82 3,91 1 0 12	56 67,67 5,82 3,91 -	56 128,6 5,82 3,91 -	56 88,65 5,82 3,91 -	56 227,98 5,51 3,61 82,51	RLE - - -

Table A.21: Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,10\}$

p	m	One Step	Add First	Add Best	Delete First	Delete Best	Cycle First	Cycle Best	BestOff	Priority 1	Priority 2	VNS1 $\alpha = 0,05$	VNS1 $\alpha = 0,1$	VNS2 $\alpha = 0,05$	VNS2 $\alpha = 0,1$	Optimal
		edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Three	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	edge # time gap-1 gap-10	Best
1	780	71 0,23 7,54 7,19	90 11,01 3,41 3,08	88 193,25 2,56 2,23	68 1,06 7,25 6,91	68 3,73 7,25 6,91	70 2,34 5,12 4,78	71 24,2 5,11 4,77	82 493,09 1,13 0,8	81 271,74 1,3 0,98	83 299,28 1,14 0,81	82 1005,55 1,18 0,85	82 986,28 1,01 0,68	81 735,91 1,19 0,86	80 1268,59 1,13 0,81	RLE - - 0,96
0,75	596	80 0,23 5,72 5,6	96 8,33 3,62 3,51	92 98,56 1,97 1,86	76 1,45 4,92 4,8	76 5,86 4,92 4,8	77 3 4,38 4,26	76 17,43 4,16 4,05	86 306,17 0,67 0,55	83 176,19 0,84 0,73	83 180,28 0,84 0,73	84 662,97 0,66 0,54	84 913,66 0,5 0,39	83 534,58 0,84 0,73	84 1310,44 0,5 0,39	RLE - - 0,46
0,5	406	91 0,23 2,89 2,85	95 5,3 2,4 2,37	95 24,51 2,33 2,3	79 1,62 1,88 1,85	79 16,45 1,83 1,8	81 2,9 1,34 1,3	83 24,83 1,27 1,23	81 198,18 0,71 0,67	82 62,95 0,83 0,79	82 91,99 0,79 0,76	81 483,67 0,68 0,65	81 590,32 0,65 0,62	81 646,85 0,65 0,62	82 490,36 0,78 0,75	RLE - 0,84 0,49

Table A.22: Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 1000, Demand $\in \{1,100\}$

p	m	One Step edge # time gap-1 gap-10	Add First edge # time gap-1 gap-10	Add Best edge # time gap-1 gap-10	Delete First edge # time gap-1 gap-10	Delete Best edge # time gap-1 gap-10	Cycle First edge # time gap-1 gap-10	Cycle Best edge # time gap-1 gap-10	BestOff Three edge # time gap-1 gap-10	Priority 1 edge # time gap-1 gap-10 Add Delete Cycle	Priority 2 edge # time gap-1 gap-10 Add Delete Cycle	VNS1 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS1 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 05$ edge # time gap-1 gap-10 Best	VNS2 $\alpha = 0, 1$ edge # time gap-1 gap-10 Best	Optimal edge # time gap-1 gap-10 Best
1	780	59 0,23 50,84 48,59	59 11,85 50,84 48,59	59 11,67 50,84 48,59	46 0,52 29,66 27,73	48 3,03 27,57 25,67	48 2,47 16,01 14,28	47 16,56 16,6 14,86	45 453,07 10,06 8,95 0 15 11	45 32,96 16,49 14,76 0 2 20	45 63,19 16,49 14,76 0 2 20	45 153,59 12,17 10,5 80,41 145,69	45 285,8 12,35 10,67 145,69	45 185,81 11,76 10,09 79,1	45 408,08 13,36 11,67 191,99	RLE - - -
0,75	591	58 0,27 41,9 39,2	59 9,05 41,81 39,11	59 17,41 41,81 39,11	49 0,46 24,33 21,96	48 2,85 18,29 16,04	49 2,18 15,11 12,92	46 16,24 11,25 9,14	45 250,13 10,49 8,38 0 14 5	45 27,96 10,12 8,03 0 1 18	45 36,85 10,12 8,03 0 1 18	44 158,09 8,23 6,17 103,88	45 142,08 10,12 8,03 -	44 213,44 8,23 6,17 131,07	44 290,93 8,23 6,17 128,06	RLE - - -
0,5	378	53 0,28 29,22 26,23	53 5,26 29,22 26,23	53 5,27 29,22 26,23	49 0,23 21,75 18,94	46 1,32 18,71 15,96	47 2,12 12,12 9,53	46 10,94 12,38 9,79	46 114,26 9,7 7,16 0 7 6	46 18,12 12,38 9,79 0 0 12	46 17,59 12,38 9,79 0 0 12	46 96,84 10,86 8,29 31,23	46 122,11 11,99 9,4 48,25	46 180,94 10,79 8,23 125,56	46 141,15 10,84 8,27 29,55	RLE - - -

Table A.23: Computational Results for 2EDPND Problem with Node Number = 40 , Coefficient= 10000, Demand $\in \{1,10\}$

