A SINGLE CHINESE POSTMAN PROBLEM WITH TWO OBJECTIVES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EZGİ EROĞLU

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

MAY 2015

Approval of the thesis:

A SINGLE CHINESE POSTMAN PROBLEM WITH TWO OBJECTIVES

submitted by **EZGİ EROĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Mevlüde Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Murat Köksalan _____
Head of Department, **Industrial Engineering**

Prof. Dr. Meral Azizoğlu _____
Supervisor, **Industrial Engineering Dept, METU**

Examining Committee Members:

Prof. Dr. Yasemin Serin _____
Industrial Engineering Dept, METU

Prof. Dr. Meral Azizoğlu _____
Industrial Engineering Dept., METU

Assoc. Prof. Dr. Esra Karasakal _____
Industrial Engineering Dept., METU

Assist. Prof. Dr. Melih Çelik _____
Industrial Engineering Dept., METU

Assoc. Prof. Dr. Ferda Can Çetinkaya _____
Industrial Engineering Dept., Çankaya University

Date: May 26, 2015

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

Name, Last : Ezgi EROĞLU

Signature :

ABSTRACT

A SINGLE CHINESE POSTMAN PROBLEM WITH TWO OBJECTIVES

Eroğlu, Ezgi
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

May 2015, 73 pages

The Chinese Postman Problem (CPP) is an arc routing problem in which a single postman serves a number of streets starting from a post office. The postman has to visit the households on each street in his route, delivering and collecting letters and then returning to the post office.

The single objective CPP minimizes the total cost of the travel. The multi-objective CPPs consider other attributes like total distance and total time, etc. In this thesis, we consider a multi-objective CPP where each street is represented by two weights, like cost and distance.

We propose a branch and bound algorithm that generates all efficient points with respect to the total cost and total distance criteria. Our algorithm benefits from the optimal solutions of the linear programming relaxations in defining the branching scheme and providing lower and upper bounds.

Our extensive computational results show that our algorithm generates the efficient solution set for large-sized problem instances in reasonable time.

Keywords: Chinese Postman Problem, Multi-Criteria, Efficient Points, Branch and Bound Algorithm

ÖZ

İKİ AMAÇLI TEK ÇİNLİ POSTACI PROBLEMİ

Eroğlu, Ezgi
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu

Mayıs 2015, 73 sayfa

Ayrıt rotalama problemlerinden biri olan Çinli Postacı Problemi, tanımlanan ağ üzerinden tüm bağlantıların tek bir postacı tarafından, mektupların dağıtılıp toplanması için ziyaret edilerek, başlangıç noktasına dönmesi problemi olarak tanımlanabilir.

Tek amaçlı Çinli Postacı Problemi, toplam seyehat maliyetini en aza indirgemektedir. Çok amaçlı Çinli Postacı Problemi ise toplam mesafe, toplam zaman vb. gibi diğer özellikleri de dikkate almaktadır. Bu tezde, her bağlantı için maliyet ve uzunluk olmak üzere iki özellik tanımlanan çok amaçlı Çinli Postacı Problemi incelenmiştir.

Toplam maliyet ve toplam mesafe kriterlerine göre tüm etkin çözümleri üreten bir dal-sınır algoritması tanımlanmıştır. Algoritmamız lineer programlama gevşetim yöntemleri kullanılarak üretilen etkin alt ve üst sınırlar kullanmaktadır.

Deneysel sonuçlarımız, büyük boyutlu problemler için etkin çözüm setinin makul sürelerde üretilebileceğini göstermiştir.

Anahtar Kelimeler: Çinli Postacı Problemi, Çok Amaçlı Karar Verme, Etkin Çözümler, Dal-Sınır Algoritması

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor Prof. Dr. Meral Azizoğlu. I am grateful to her for her guidance, advice, criticism and encouragements throughout my M.S. study. It was a great experience working with her.

I would like to express my deepest thanks to my parents, Efser Eroğlu and Ersin Eroğlu, for their endless patience and supports.

Lastly, I am grateful to my brother, Ergin Eroğlu, for his unconditional support and guidance. I would like to express my appreciation to him for adding value to my thesis from a computer engineer perspective.

To My Family

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The arc routing problems (ARPs) entail determining a minimum-cost traversal of a specified subset of arcs on a graph. The earliest study related to the ARPs is the Königsberg Bridge Problem. The problem had been solved by the Swiss mathematician Leonhard Euler in 1736, who aimed to determine whether there exists a closed walk traversing each of the seven bridges given in Figure 1, once and exactly once.

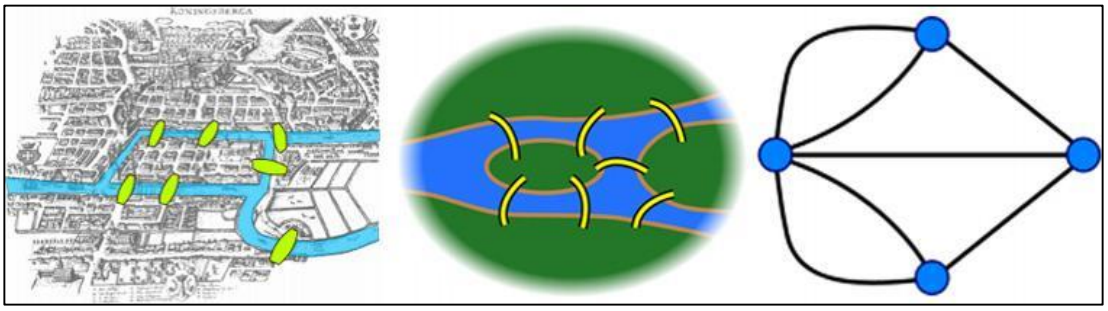


Figure 1: The Seven Bridges of Königsberg

As can be seen from Figure 1, the bridges of Königsberg are represented as arcs and each arc joins a pair of the four junction points (nodes) which can be considered as islands and shores.

The Königsberg Bridge Problem is only concerned with the existence and determination of a closed walk. The Chinese Postman Problem (CPP) takes its origin from the Königsberg Bridge Problem and deals with situations where such a solution

does not exist. The CPP determines a minimum length walk covering each segment at least once. The problem was initially posed by a mathematician at the Shangtun Normal College in 1962, called Meigu Guan (or Kwan Mei-Ko). Guan (1962) simply states the problem as follows: Leaving from his post office, a postman needs to visit the households on each block in his route, delivering and collecting letters, and then returns to the post office. He would like to cover this route by traveling the minimum possible distance.

Many mathematicians and operations researchers model real life situations as Chinese Postman Problem. Some noteworthy real applications are postal delivery, parcel services, garbage collection, milk delivery, snow clearance, street sweeping services, routing of salt trucks and inspection of streets for maintenance.

The CPP has been studied on various types of graphs including undirected, directed, mixed and windy graphs. Operations researchers have studied for a long time the structure of classical CPP on these graphs and proposed workable solutions. The classical CPP assumes that each arc is represented by a single weight and the objective is to determine the tour with the minimum total weight.

The multi-objective CPPs assume that each arc is attributed by two weights, like cost and distance, distance and priority, cost and time, etc. Many practical implications might require consideration of both weights, thereby two objectives. For example, the total distance of the route should be minimized subject to the constraint that the total cost of the route is below a threshold value. Moreover, the decision maker may like to see the solutions over which he/she could make a trade-off. The concern might be the amount compromise that should be made over total distance to reduce the total cost to some specified level. Such a concern would be answered if all non-dominated solutions with respect to two objectives are known.

Despite its practical importance, the literature on the multi-objective CPP is quite scarce. To the best of our knowledge, there is a unique study by Prakash et al. (2009), which considers the CPP with two objectives. Their work is limited to the complete enumeration of feasible solutions from which they select the non-dominated ones.

Recognizing this important gap in the literature, we propose a branch and bound algorithm for directed CPP that generates all non-dominated (efficient) points with respect to two objectives. Our algorithm benefits from the optimal solutions of the Linear Programming Relaxations (LPRs). The optimal LPR solutions are used to define our branching scheme and to provide lower and upper bounds to restrict the size of the search.

The rest of the thesis is organized as follows: In Chapter 2, we review the literature on the CPPs and its related problems. Chapter 3 defines our problem and presents the models of our bi-objective problem. In Chapter 4, we settle the complexity of the problem and propose a branch and bound algorithm for its solution. Chapter 5 discusses the results of our extensive computational experiment. We conclude in Chapter 6 by pointing out our main findings and suggestions for future research.

CHAPTER 2

THE BASICS AND LITERATURE SURVEY

In this section, we first review the single-objective Chinese Postman Problem (CPP). The review covers the basics of the problem and related literature. We then discuss the multi-objective Chinese Postman Problem. Finally, we review the related literature on some variants of the CPP.

2.1 The Basics of the Chinese Postman Problem

The Chinese Postman Problem is an arc routing problem in which a single postman serves a number of streets from a post office. The problem is represented by a graph, G = (N, A), where the arcs (A) represent the streets and the nodes (N) represent the junction points and the post office.

The classical CPP assumes that each arc is represented by a nonnegative cost function and the objective is to find the minimum cost closed walk passing through every arc of graph at least once. This problem is firstly studied by Guan (1962).

The CPP problem has several real-life applications including post-delivery, parcel services, garbage collection, milk delivery, snow clearance, street sweeping services, routing of salt trucks and inspection of streets for maintenance (Pearn and Chiu 2005, Corberan and Prins 2010).

In order to find minimum total cost tour, the property of unicursality is used, which means there is a closed walk in a connected graph covering each node at least once and each arc exactly once. In the study of Ford and Fulkerson (1962), the sufficient conditions for a connected graph to be unicursal are stated. For the directed case, the number of entering and leaving arcs to each node must be equal.

The CPP is studied on various types of graphs in the literature including undirected, directed, mixed and windy versions. Since the directed and undirected versions of the problem are solvable in polynomial time, most of the researchers recently focus on mixed and windy variants (Corberan and Prins 2010).

Two formulations are proposed for the undirected CPP in Eiselt et al. (1995). The first formulates the problem as a perfect matching problem which aims to turn the graph G into G′ such that all nodes of it have an even degree. By introducing the shortest paths which correspond to the optimal matching solution, graph G′ can be obtained from G. The second algorithm, on the other hand, is based on Edmonds' blossom inequalities (see Edmonds and Johnson 1965) and it depends on the density of arcs in graph G.

In the directed case, the problem has a feasible solution provided that the graph is strongly connected, which means there must be a directed path between every pair of nodes, and the arc weights are nonnegative. The optimal solution is found by a polynomial time algorithm as proposed in Eiselt et al (1995). It is shown that a minimum cost unicursal (Eulerian) graph from G can be constructed by solving transportation problem where the optimal decision variable values represent the number of extra times each arc has to be traversed. In the study by Thimbleby (2003), complete and executable Java code is presented in order to solve directed version of the CPP.

Windy graphs are undirected graphs despite the fact that the cost of traversing an arc differs with respect to the direction of travel. Windy postman problem (WPP), which was firstly introduced by Minieka (1979), consists of finding a minimum cost closed walk passing through every arc of G at least once. It is shown that the WPP is NP-hard by Brucker (1981). When the graph is unicursal, Win (1989) shows that the problem can be solved in polynomial time. Grötschel and Win (1992) also describe the cutting plane algorithm to solve the WPP, which is considered to be the best known exact method up to now.

Mixed graphs contain both undirected and directed arcs and consist of determining a least-cost traversal through every arc of G at least once. Papadimitriou (1976) shows that finding an optimal CPP in mixed graphs is NP-hard. However, if the graph is even (the total number of undirected and directed arcs incident to each of its nodes is even),

then the problem is polynomially solvable (Edmonds and Johnson, 1973). To find the optimal solution, some authors, including Christofides et al. (1984) and Nobert and Picard (1991), have used integer linear programming formulations. The branch and cut algorithm is proposed by Corberan et al. (2012) to solve large size Mixed CPPs.

Once the Eulerian (Unicursal) graph is determined, than a Eulerian circuit, which is a closed path crossing each arc at least once, has already been constructed. For the undirected version of the CPP, Fleury's algorithm was initially proposed (see Kaufmann, 1967) and reported as inefficient in Eiselt et al (1995). As an alternative method, which is also more efficient, End-pairing Algorithm was proposed by Edmonds and Johnson (1973). For the directed Eulerian graph, the circuit can be drawn by simply adapting the Fleury's Algorithm (Christofides, 1975).

2.2 Multi-objective Chinese Postman Problem

To the best of our knowledge, there is a unique study that considers multiple objectives for the CPP. This study is due to the conference proceeding authored by Prakash et al. (2009).

They present a heuristic procedure that finds a set of feasible solutions giving higher priority to the first objective. They state that the performance of their procedure depends on the choice of the priorities and the procedure may return a single efficient point or a subset of all efficient solutions or a set of all efficient points. They implement their procedure via a graph residing 9 nodes and 12 arcs.

2.3 Extensions

Until now, we have focused on the classical CPP and the multi-objective CPP. In the literature, different versions of the Chinese Postman Problem have been studied by a number of researchers. In this section, we focus on the prominent extensions of the CPP.

2.3.1 Generalized Chinese Postman Problem

The Generalized CPP is the arc routing version of the Generalized Travelling Salesman Problem and entails finding a minimum cost tour traversing at least one of the arcs of

each predefined arc subset. This problem is initially defined and proved to be strongly NP-hard by Dror and Haouari (2000).

2.3.2 Rural Postman Problem (RPP)

In Rural Postman Problem, there is a set of arcs which has to be serviced by a postman, and a set of non-required arcs that do not have to be served but may be used for travelling between nodes. According to the study by Eiselt et al. (1995), the RPP is commonly associated with mail delivery in rural areas contrary to the CPP which is more likely to arise in urban settings. The undirected and directed RPPs are polynomially solvable if the required arcs induce a connected graph. Otherwise, the problem is strongly NP-hard.

The RPP has received a lot of attention in the last decade and different versions of the problem has been studied by a number of researchers. Corberan and Prins (2010) present an overview of the most relevant variants of the problem.

2.3.3 Hierarchical Chinese Postman Problem

The Hierarchical CPP (HCPP) is another strongly NP-hard variant of the CPP. In the HCPP, a precedence relation is established within the different arc subsets. Accordingly, if subset i precedes subset j, then the arcs of subset i must be travelled before the arcs of j. Although HCPP is strongly NP-hard in general, Dror et al. (1987) proposes a polynomially solvable special case when there is a liner precedence relation, which divides the elements of arc set into pairwise disjoint classes with a certain priority index indicating which class of arcs has to be traversed first, second, etc. In the study of Cabral et al. (2004), the HCPP is transformed into the RPP, which also includes the results related to the computational value of the transformation. Several different algorithms that require less computational efforts and other variants of the HCPP are discussed in Ghiani and Improta (2000) and Korteweg and Volgenant (2006).

2.3.4 k-Person Chinese Postman Problem (k-CPP)

The k-CPP is a Capacitated Arc Routing Problem which can be considered as a multiple-vehicle extension of the classical CPP with $k \geq 2$ postmen. The problem is briefly defined by Pearn (1994) as finding a set of k postman tours so that:

- each postman route begins and ends at a common node called the central depot,
- each arc in G must be serviced by exactly one postman,
- all the k postmen must be involved in the delivery service,
- and the total distance traveled is minimized.

The k-CPP is strongly NP-hard by Thomassen (1997). However, there are several polynomially solvable special cases of the problem. For the totally undirected version of the problem Benavent et al. (1992) propose an algorithm which solves this class of k-CPP optimally. For the k-CPP on totally directed networks, the algorithm proposed for the undirected case can be easily extended. Pearn (1991) shows that the other two versions of the k-CPP, mixed but even-and-symmetric networks and windy networks with symmetric cycles are also polynomially solvable.

The Min-Max k-CPP is an extension of the classical k-CPP. The problem entails minimizing the cost of the tour with the maximum cost instead of minimizing the sum of all tour costs. Frederickson et al. (1978) show that the problem is NP-hard in the strong sense. Several heuristic algorithms are developed for this problem and related computational results are reported in Ahr and Reinelt (2002). Another extension, Minimum Absolute Deviance (MAD) k-CPP, was introduced by Degenhardt (2004). The aim of the problem is to minimize the sum of all deviations from an allowed service time for a single postman.

2.3.5 Maximum Benefit Chinese Postman Problem (MBCPP)

The Maximum Benefit Chinese Postman Problem (MBCPP) was initially introduced by Malandraki and Daskin (1993) on directed graphs. In this problem, each arc is associated with a service and deadhead costs and a list of nonincreasing profits. A prize can be collected from each arc multiple times, but the size of the prize decreases as the

number of traversals increase. The authors present the problem as a minimum cost flow problem with subtour elimination constraints. They model a branch and bound based algorithm and solve small size instances on a graph with up to 25 nodes and 43 arcs. The heuristic algorithms of constructive type and local search procedures for the directed version are presented in the study by Pearn and Chiu (2005). The computational results are based on instances with 30 nodes and 780 arcs.

Pearn and Wang (2003) consider the MBCPP on undirected graphs. It is shown that the Rural Postman Problem is a special case of the problem; hence, it is strongly NP-hard. The authors present a heuristic solution based on expanding the network, finding a minimum spanning tree followed by the construction of a minimum cost matching. The procedure is illustrated on a graph with 15 nodes and 26 arcs.

In a latter study by Corberan et al. (2013), Integer Programming formulation is proposed for the undirected version of the MBCPP. They propose a branch-and-cut algorithm and show that the algorithm is capable of solving instances with up to 1000 nodes and 3000 arcs.

2.3.6 Chinese Postman Problem with Time Windows (CPPTW)

The CPP with time windows (CPPTW) is initially proposed by Aminu and Eglese (2006). The problem assumes that the traversal starts and finishes at the depot node. Moreover, time window constraints are introduced at each arc in order to restrict the earliest and latest time for completing the service of arcs. To solve the CPPTW, the authors propose two different models and present the computational results. The first model solves the problem directly by constraint programming, and the second one transforms the problem into an equivalent node routing problem with time windows.

2.3.7 Prize-Collecting Rural Postman Problem (PRPP)

Recall that contrary to the CPP, in the Rural Postman Problem, only a subset of the arcs of the graph, i.e., required arcs, have to be traversed at least once. The Prize-Collecting RPP can be considered as the arc routing version of the Prize-Collecting Travelling Salesman Problem. The problem is introduced by Aràoz et al. (2006) and entails finding a cycle that passes through the distinguished node (depot) while

maximizing the value of the arcs. The value of the arc corresponds to the total servicing profit (which can be collected only once) minus the traversal costs (which is paid for each traversal). In a later study, Aràoz et al. (2009) present a two-phase LP-based cutting plane algorithm to solve the PRPP. The first phase transforms the problem into the RPP to obtain a feasible solution and the second phase continues with a Branch and Cut Algorithm. The computational results show that the algorithm can solve 94 out of 118 instances with up to 100 nodes and 22 subset of connected arcs.

2.3.8 Profitable Arc Tour Problem (PATP)

The Profitable ATP is defined by Feillet et al. (2005), and considers a similar version of the PRPP where the prizes can be collected multiple times from a given arc in a directed graph. The problem entails finding a set of cycles in the graph so as to maximize the collection of profit minus travel costs subject to the constraints limiting the number of times that profit is available on arcs and the maximum length of the cycles. The authors assume that the prize to be collected from a given arc is constant over time and there are unlimited number of vehicles, not necessarily related to any depot but with a limit on their tour length. They solve the problem by Branch-and-Price algorithm that is capable of solving instances with up to 30 nodes and hundreds of arcs. Euchi et al. (2011) study the PATP under the assumption that the length and the cost of an arc are independent. They propose tabu search and variable neighborhood search algorithms and report on their satisfactory performances.

CHAPTER 3

PROBLEM DEFINITION AND THE RELATED MODELS

Consider a directed graph G = (N, A) that consists of a set of A arcs and N nodes. Arc *(i,j)* establishes a connection between nodes *i* and *j*, and is characterized by two weights, c_{ij} and d_{ij}. c_{ij} may stand for the cost of traversing arc (*i,j*) whereas d_{ij} is its travel time or distance.

The main decision of the problem is explained via the variable X_{ij} as follows:

$$X_{ij} = \text{number of times arc } (i,j) \text{ is traversed}$$

The objective of the problem is to minimize the total cost that is expressed as

$$\sum_{(i,j)\in A} c_{ij} \times X_{ij} \tag{1}$$

The constraints are stated below:

i. Each arc should be traversed at least once:

$$X_{ij} \geq 1 \text{ and integer} \qquad \forall\,(i,j) \in A \tag{2}$$

ii. The flow to a node should be conserved that is for each entering arc, there should be a departing counterpart.

$$\sum_{i \mid (i,j)\in A} X_{ij} = \sum_{i \mid (j,i)\in A} X_{ji} \qquad for\ i \in N \tag{3}$$

The problem of minimizing (1) subject to the constraint sets (2) and (3) is a minimum cost directed Chinese Postman Problem. The CPP is an easy to solve problem.

The rest of this chapter is organized as follows: In Section 3.1, we discuss the solution procedures on the single objective CPP. Section 3.2 discusses the constrained CPP. In section 3.3 we discuss, the efficient points and their generation.

3.1 Solution Procedures for the Chinese Postman Problem

In the directed case, a minimum cost Eulerian graph from G(N, A) can be found by solving a minimum cost flow problem where the flow on each arc has to be at least 1.

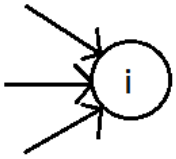
For a solution to exist, the graph should be strongly connected, i.e., there must be a directed path between every pair of nodes.

It is shown by Eiselt et al. (1995) that a least cost Eulerian graph can be constructed by solving a transportation problem.

In order to introduce the procedure, the following definitions are needed:

i. $d_i^+ = \begin{cases} \text{number of incoming arcs to node i} \\ \text{indegree of node i} \end{cases}$

According to the following subgraph $d_i^+ = 3$



ii. $d_i^- = \begin{cases} \text{number of outgoing arcs from node i} \\ \text{outdegree of node i} \end{cases}$

According to the following subgraph $d_i^- = 4$

iii. The node is called balanced if $d_i^+ = d_i^-$.

iv. The sets below are defined for the unbalanced nodes:

$$I = \{i \mid d_i^+ > d_i^-\}$$

$$J = \{i \mid d_i^+ < d_i^-\}$$

Set I represents the supply nodes since indegree is greater than outdegree, whereas Set J represents the demand nodes in the transportation problem.

The parameters and decision variable used in the model of transportation problem are defined as follows:

$SP_{ij} =$ length of the shortest path between node i and node j

$S_i = d_i^+ - d_i^-, \quad D_j = d_j^- - d_j^+$

$X_{ij} =$ number of extra times arc (i,j) is traversed

$$Z_T = Min \sum_{i \in I} \sum_{j \in J} SP_{ij} \times X_{ij}$$

subject to

$$\sum_{j \in J} X_{ij} = S_i \qquad \forall\, i \in I$$

$$\sum_{i \in I} X_{ij} = D_j \qquad \forall\, j \in J$$

$$X_{ij} \geq 0$$

As stated in Eiselt et al. (1995), an optimal solution to the above model, which is denoted by Z_T^*, gives an optimal solution to the CPP. As X_{ij} is defined as the number of extra times arc(i, j) is traversed, the objective function value of the CPP is $Z_T^* + \sum c_{ij}$.

We simply refer to the model (1) through (3) as P_C. P_D, on the other hand, is the problem where the objective function of P_C is changed as follows:

$$Minimize \sum_{(i,j) \in A} d_{ij} \times X_{ij}$$

3.2 The Constrained Chinese Postman Problem

Consider the following constraint that gives an upper bound on the total distance travelled:

$$\sum_{(i,j)\in A} d_{ij} \times X_{ij} \leq k \tag{4}$$

where *k* is a fixed upper bound value on the total distance travelled.

The problem of minimizing (1) subject to the constraint sets (2), (3) and (4) is a single constrained CPP.

We hereafter refer to the single constrained CPP that uses *k* as an upper bound of the single constraint as P_C,*k*.

Feasible and optimal solutions to the (P_C) and (P_C,*k*) are illustrated via the following 25 nodes and 43 arcs problem which is taken from Malandraki and Daskin (1993). We hereafter refer to the instance as I. Our example problem instance is represented in Figure 2.

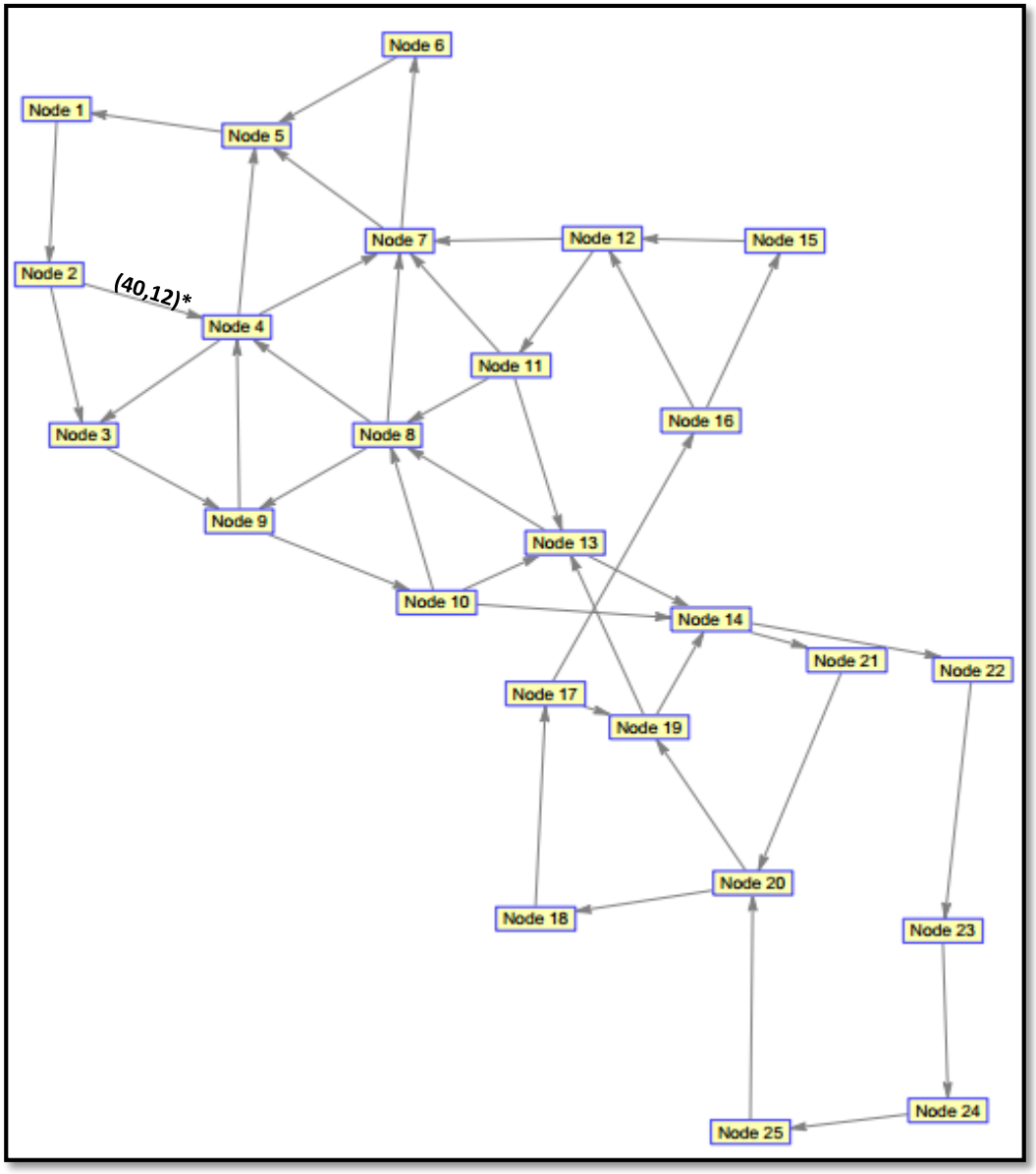


Figure 2: Example Instance I

The data related to the arc weights, i.e. (c_{ij}, d_{ij}) values, are given in Table 1 below. As an example, weight vector of arc (2, 4) is $(c_{2,4}, d_{2,4}) = (40, 12)$.

Table 1: Data Related to Arc Weights

(i, j)	(c_{ij}, d_{ij})	(i, j)	(c_{ij}, d_{ij})	(i, j)	(c_{ij}, d_{ij})	(i, j)	(c_{ij}, d_{ij})
(1,2)	(40,40)	(8,4)	(50,50)	(12,7)	(90,90)	(18,17)	(30,30)
(2,3)	(30,30)	(8,7)	(30,30)	(12,11)	(20,20)	(19,13)	(40,40)
(2,4)	(40,12)	(8,9)	(60,60)	(13,8)	(70,70)	(19,14)	(70,70)
(3,9)	(40,40)	(9,4)	(70,70)	(13,14)	(70,21)	(20,18)	(30,30)
(4,3)	(40,12)	(9,10)	(60,60)	(14,21)	(20,80)	(20,19)	(30,30)
(4,5)	(30,30)	(10,8)	(60,60)	(14,22)	(40,40)	(21,20)	(20,70)
(4,7)	(50,50)	(10,13)	(60,9)	(15,12)	(40,20)	(22,23)	(30,30)
(5,1)	(50,50)	(10,14)	(30,36)	(16,12)	(40,60)	(23,24)	(30,30)
(6,5)	(50,15)	(11,7)	(80,80)	(16,15)	(40,20)	(24,25)	(80,20)
(7,5)	(50,50)	(11,8)	(70,70)	(17,16)	(40,40)	(25,20)	(70,20)
(7,6)	(30,9)	(11,13)	(30,30)	(17,19)	(30,30)		

The optimal solution to *(P_C)* is stated below:

$X^*_{1,2} = 5,\ X^*_{2,3} = 4,\ X^*_{5,1} = 5,\ X^*_{7,5} = 3,\ X^*_{3,9} = 5,\ X^*_{9,10} = 5,\ X^*_{12,11} = 3,$
$X^*_{16,12} = 3,\ X^*_{10,14} = 3,\ X^*_{13,14} = 2,\ X^*_{17,16} = 4,\ X^*_{18,17} = 5,\ X^*_{20,18} = 5,$
$X^*_{21,20} = 5, X^*_{14,21} = 5$

and all other $X^*_{i,j}$ values are equal to 1.

The optimal total cost, which is denoted by Z_C^*, is equal to:

$$Z_C^* = \sum_{(i,j)\in A} c_{ij} \times X_{ij}^*$$

$$\begin{aligned} Z_C^* = {} & 40\times 5 + 30\times 4 + 40\times 1 + 40\times 5 + 40\times 1 + 30\times 1 + 50\times 1 + 50\times 5 \\ & + 50\times 1 + 50\times 3 + 30\times 1 + 50\times 1 + 30\times 1 + 60\times 1 + 70\times 1 \\ & + 60\times 5 + 60\times 1 + 60\times 1 + 30\times 3 + 80\times 1 + 70\times 1 + 30\times 1 \\ & + 90\times 1 + 20\times 3 + 70\times 1 + 70\times 2 + 20\times 5 + 40\times 1 + 40\times 1 \\ & + 40\times 3 + 40\times 1 + 40\times 4 + 30\times 1 + 30\times 5 + 40\times 1 + 70\times 1 \\ & + 30\times 5 + 30\times 1 + 20\times 5 + 30\times 1 + 30\times 1 + 80\times 1 + 70\times 1 \end{aligned}$$

$Z_C^* = 3700$

The optimal solution to (P_D) is stated below:

$X^*_{1,2} = 5, X^*_{2,4} = 4, X^*_{4,3} = 4, X^*_{5,1} = 5, X^*_{6,5} = 3, X^*_{7,6} = 3, X^*_{3,9} = 5,\ X^*_{9,10} = 5,\ X^*_{12,11} = 3,\ X^*_{15,12} = 3,\ X^*_{16,15} = 3,\ X^*_{10,13} = 3,\ X^*_{13,14} = 4,\ X^*_{17,16} = 4,\ X^*_{18,17} = 5,\ X^*_{20,18} = 5,\ X^*_{22,23} = 5,\ X^*_{14,22} = 5,\ X^*_{23,24} = 5,\ X^*_{24,25} = 5,\ X^*_{25,20} = 5$

where all other $X^*_{i,j}$ values are equal to 1.

The optimal total distance, which is denoted by Z^*_D, is equal to:

$$Z^*_D = \sum_{(i,j)\in A} d_{ij} \times X^*_{ij}$$

$$\begin{aligned} Z^*_D = {} & 40\times5 + 30\times1 + 12\times4 + 40\times5 + 12\times4 + 30\times1 + 50\times1 + 50\times5 \\ & + 15\times3 + 50\times1 + 9\times3 + 50\times1 + 30\times1 + 60\times1 + 70\times1 \\ & + 60\times5 + 60\times1 + 9\times3 + 36\times1 + 80\times1 + 70\times1 + 30\times1 \\ & + 90\times1 + 20\times3 + 70\times1 + 21\times4 + 80\times1 + 40\times5 + 20\times3 \\ & + 60\times1 + 20\times3 + 40\times4 + 30\times1 + 30\times5 + 40\times1 + 70\times1 \\ & + 30\times5 + 30\times1 + 70\times1 + 30\times5 + 30\times5 + 20\times5 + 20\times5 \end{aligned}$$

$Z^*_D = 3755$

Now, assume *k* value of (P_C, k) is 3890. The constraint $\sum_{(i,j)\in A} d_{ij} \times X^*_{ij} \leq 3890$ is not satisfied for the optimal X^*_{ij} of the (P_C, k) problem since $\sum_{(i,j)\in A} d_{ij} \times X^*_{ij} = 3917$.

The optimal solution to *(P_C, k)* problem with *k* value of 3890 is found as follows:

$X^*_{1,2} = 5,\ X^*_{2,3} = 4,\ X^*_{5,1} = 5,\ X^*_{6,5} = 3,\ X^*_{7,6} = 3,\ X^*_{3,9} = 5,\ X^*_{9,10} = 5,\ X^*_{12,11} = 3,\ X^*_{16,12} = 3,\ X^*_{10,14} = 3,\ X^*_{13,14} = 2,\ X^*_{17,16} = 4,\ X^*_{18,17} = 5,\ X^*_{20,18} = 5, X^*_{21,20} = 5, X^*_{14,21} = 5$

where all other $X^*_{i,j}$ values are equal to 1.

The optimal total cost, which is denoted by $(Z_C, k)^*$, is equal to:

$$Z^*_{C,k} = \sum_{(i,j)\in A} c_{ij} \times X^*_{ij}$$

$$Z^*_{C,k} = 40 \times 5 + 30 \times 4 + 40 \times 1 + 40 \times 5 + 40 \times 1 + 30 \times 1 + 50 \times 1 + 50 \times 5$$
$$+ 50 \times 3 + 50 \times 1 + 30 \times 3 + 50 \times 1 + 30 \times 1 + 60 \times 1 + 70 \times 1$$
$$+ 60 \times 5 + 60 \times 1 + 60 \times 1 + 30 \times 3 + 80 \times 1 + 70 \times 1 + 30 \times 1$$
$$+ 90 \times 1 + 20 \times 3 + 70 \times 1 + 70 \times 2 + 20 \times 5 + 40 \times 1 + 40 \times 1$$
$$+ 40 \times 3 + 40 \times 1 + 40 \times 4 + 30 \times 1 + 30 \times 5 + 40 \times 1 + 70 \times 1$$
$$+ 30 \times 5 + 30 \times 1 + 20 \times 5 + 30 \times 1 + 30 \times 1 + 80 \times 1 + 70 \times 1$$

$$Z^*_{C,k} = 3760$$

The travel distance constraint is also satisfied:

$$\sum_{(i,j)\in A} d_{ij} \times X^*_{ij} = 3865 < 3890$$

Note that $(Z_C, k)^*$ is bigger than Z^*_C, i.e. $Z^*_C = 3700 < 3760 = Z^*_{C,k}$

3.3 Efficient Points

A solution *u* is called efficient if there is no other solution *v*, having $Z^v_C \leq Z^u_C$ and $Z^v_D \leq Z^u_D$ with at least one of these inequalities holding strictly. The resulting objective function vector (Z^u_C, Z^u_D) is said to be efficient.

If there exists a solution v such that $(Z^u_C, Z^u_D) \leq (Z^v_C, Z^v_D)$, then the solution *v* dominates solution u and objective vector (Z^v_C, Z^v_D) dominates the objective vector (Z^u_C, Z^u_D). We thereafter refer to the objective vector as efficient point.

In Section 3.3.1, we present the properties of the efficient points. In Section 3.3.2, we discuss the generation of a single efficient point. Section 3.3.3 presents efficient set generation. Section 3.3.4 illustrates the efficient point and efficient set generation on an example instance.

3.3.1 Properties of All Efficient Points

Property 1. In all feasible solutions; hence, efficient solutions, for any node *i* whose indegree is 1, i.e., $d^+_i = 1$, $x_{ji} \geq d^-_i$ where arc (*j*, *i*) is incident to node *i*.

Proof. There are at least d_i^- inflows to node *i*; hence, at least d_i^- outflows from node *i*. The only way of realizing outflows is traversing arc (*j*, *i*), at least d_i^- times. It follows that $x_{ji} \geq d_i^-$.

□

Property 2. In all efficient solutions, for any node *i* whose outdegree is 1, i.e. $d_i^- = 1$, $x_{ij} \geq d_i^+$ for the arc (*i*, *j*) incident to node *i*.

Proof. Similar to that of Property 1; hence, it is omitted.

Property 3. An arc incident to node *r* is traversed exactly once if node *r* is not on any shortest path that connects $i \in I$ to j $\in J$.

Proof. In order to solve single objective CPP, we model the problem as a transportation problem which traverses any arc incident to a balance node *r* exactly once.

This follows that if any balanced node is not on any shortest path between $i \in I$ to j $\in J$, relative to both arc weights, i.e., c_{ij} *and* d_{ij} values, then it is traversed exactly once in all efficient solutions.

□

We now illustrate the implementation of Property 3.

Example. Consider a 36-nodes network with the following arcs given in Table 2 below (taken from the official website of University of Brescia, Department of Information Engineering). We hereafter refer to the instance as II.

Note that nodes 2, 3, 7, 12, 13, 16, 18, 20, 30 and 34 are balanced as their indegrees are equal to their outdegrees. The nodes 6, 8, 9, 10, 11, 24, 25, 31, 32, 35 and 36 are supply nodes, as they have more incoming arcs than outgoing arcs, i.e., indegree – outdegree ($\boldsymbol{d_i^+ - d_i^-}$) is positive. The remaining nodes 1, 4, 5, 14, 15, 17, 19, 21, 22, 23, 26, 27, 28, 29 and 33 are demand nodes as they have more outgoing arcs than incoming arcs, i.e., indegree – outdegree is negative. 11 supply nodes require shortest path computations with 15 demand nodes. All shortest paths, which equal to 165 ($= 11 \times 15$) with respect to each arc weight, are tabulated in APPENDIX A.

Table 2: The Data Related to Instance II

i	Incident Arcs	$d_i^+ - d_i^-$
1	(1,6) (1,7) (1,25) (19,1)	-2
2	(2,3) (2,9) (20,2) (27,2)	0
3	(3,8) (3,12) (2,3) (23,3)	0
4	(4,8) (4,9) (4,24) (26,4)	-2
5	(5,6) (5,8) (5,25) (7,5)	-2
6	(6,29) (1,6) (5,6) (7,6) (8,6)	3
7	(7,5) (7,6) (1,7) (16,7)	0
8	(8,6) (3,8) (4,8) (5,8)	2
9	(9,29) (2,9) (4,9) (24,9) (27,9)	3
10	(10,31) (13,10) (28,10) (29,10)	2
11	(11,29) (14,11) (16,11) (17,11) (18,11)	3
12	(12,31) (12,32) (3,12) (14,12)	0
13	(13,10) (13,18) (15,13) (17,13)	0
14	(14,11) (14,12) (14,16) (19,14)	-2
15	(15,13) (15,16) (15,18) (17,15)	-2
16	(16,7) (16,11) (14,16) (15,16)	0
17	(17,11) (17,13) (17,15) (18,17) (36,17)	-1
18	(18,11) (18,17) (13,18) (15,18)	0
19	(19,1) (19,14) (19,21) (20,19)	-2
20	(20,2) (20,19) (26,20) (27,20)	0
21	(21,22) (21,25) (21,34) (19,21)	-2
22	(22,23) (22,26) (22,27) (21,22)	-2
23	(23,3) (23,24) (23,31) (22,23)	-2
24	(24,9) (4,24) (23,24) (26,24)	2
25	(25,33) (1,25) (5,25) (21,25) (30,25)	3
26	(26,4) (26,20) (26,24) (22,26)	-2
27	(27,2) (27,9) (27,20) (22,27) (36,27)	-1
28	(28,10) (28,32) (28,36) (29,28)	-2
29	(29,10) (29,28) (29,32) (29,36) (6,29) (9,29) (11,29)	-1
30	(30,25) (30,35) (33,30) (34,30)	0
31	(31,33) (10,31) (12,31) (23,31) (32,31)	3
32	(32,31) (12,32) (28,32) (29,32)	2
33	(33,30) (33,34) (33,35) (33,36) (25,33) (31,33)	-2
34	(34,30) (34,35) (21,34) (33,34)	0
35	(35,36) (30,35) (33,35) (34,35)	2
36	(36,17) (36,27) (28,36) (29,36) (33,36) (35,36)	2

For the 25 node pairs given in Table 3 below, the paths differ from each other with respect to arc weight set. For the remaining part given in Appendix, shortest paths do not change in terms of c_{ij} and d_{ij} values.

Table 3: Shortest Paths according to each arc weight

[i, j]	**Shortest Paths wrt c_{ij}**	**Shortest Paths wrt d_{ij}**
[6 - 5]	6-29-36-17-15-16-7-5	6-29-36-27-20-19-1-7-5
[8 - 5]	8-6-29-36-17-15-16-7-5	8-6-29-36-27-20-19-1-7-5
[9 - 5]	9-29-36-17-15-16-7-5	9-29-36-27-20-19-1-7-5
[10 - 5]	10-31-33-36-17-15-16-7-5	10-31-33-36-27-20-19-1-7-5
[10 - 28]	10-31-33-36-27-9-29-28	10-31-33-36-17-11-29-28
[10 - 29]	10-31-33-36-27-9-29	10-31-33-36-17-11-29
[11 - 5]	11-29-36-17-15-16-7-5	11-29-36-27-20-19-1-7-5
[24 - 5]	24-9-29-36-17-15-16-7-5	24-9-29-36-27-20-19-1-7-5
[25 - 5]	25-33-36-17-15-16-7-5	25-33-36-27-20-19-1-7-5
[25 - 28]	25-33-36-27-9-29-28	25-33-36-17-11-29-28
[25 - 29]	25-33-36-27-9-29	25-33-36-17-11-29
[31 - 5]	31-33-36-17-15-16-7-5	31-33-36-27-20-19-1-7-5
[31 - 28]	31-33-36-27-9-29-28	31-33-36-17-11-29-28
[31 - 29]	31-33-36-27-9-29	31-33-36-17-11-29
[32 - 5]	32-31-33-36-17-15-16-7-5	32-31-33-36-27-20-19-1-7-5
[32 - 28]	32-31-33-36-27-9-29-28	32-31-33-36-17-11-29-28
[32 - 29]	32-31-33-36-27-9-29	32-31-33-36-17-11-29
[35 - 5]	35-36-17-15-16-7-5	35-36-27-20-19-1-7-5
[35 - 28]	35-36-27-9-29-28	35-36-17-11-29-28
[35 - 29]	35-36-27-9-29	35-36-17-11-29
[35 - 33]	35-36-27-9-29-10-31-33	35-36-27-20-19-1-25-33
[36 - 5]	36-17-15-16-7-5	36-27-20-19-1-7-5
[36 - 28]	36-27-9-29-28	36-17-11-29-28
[36 - 29]	36-27-9-29	36-17-11-29
[36 - 33]	36-27-9-29-10-31-33	36-27-20-19-1-25-33

It is seen that the balanced nodes 2, 3, 12, 13, 18, 30 and 34 do not appear in any one of the shortest paths with respect to c_{ij} and d_{ij} values. Hence, those 7 nodes are eliminated leaving a reduced problem with 29 nodes. The arcs that are incident to the eliminated nodes are (2,3), (2,9), (20,2), (27,2), (3,8), (3,12), (12,32), (12,31), (14,12), (13,18), (13,10), (23,3), (15,13), (18,17), (18,11), (15,18), (30,35), (33,30), (34,35), (34,30), (33,34), (21,34) and (30,25).

All flow values are '1' at optimality for the eliminated nodes and the flow balance relations for the remaining 29 (=36-7) nodes are modified as in the Table 4 below.

Note that some right hand side values of the relations are not '0', as they consider the flow values at the eliminated arcs.

For example, the flow balance equation belonging to node 8 is originally as follows:

$$X_{8,6} = X_{3,8} + X_{4,8} + X_{5,8}$$

Since $X_{3,8} = 1$ for the eliminated arc (3, 8), the new relation is presented as follows:

$$X_{8,6} = 1 + X_{4,8} + X_{5,8}$$

Similarly, for node 15, the original relation is

$$X_{15,16} + X_{15,13} + X_{15,18} = X_{17,15}$$

Nodes 13 and 18 are eliminated together with the incident arcs (15, 13) and (15, 18). As the eliminated arcs receive value '1', i.e., $X_{15,13} = X_{15,18} = 1$, and the resulting flow relation becomes

$$X_{15,16} + 2 = X_{17,15}$$

Using the result of Property 3, we eliminate a balanced node r, if it does not appear on any shortest path connecting i ∈ I to j ∈ J, relative to both arc weights. Eliminating a node implies setting the flows of all arcs incident to node r to '1'.

Table 4: Flow Balance Equations of the Remaining Nodes

#	**Flow Balance Equations**
1	$X_{1,6} + X_{1,7} + X_{1,25} - X_{19,1} = 0$
4	$X_{4,24} + X_{4,8} + X_{4,9} - X_{26,4} = 0$
5	$X_{5,6} + X_{5,8} + X_{5,25} - X_{7,5} = 0$
6	$X_{6,29} - X_{1,6} - X_{5,6} - X_{7,6} - X_{8,6} = 0$
7	$X_{7,5} + X_{7,6} - X_{1,7} - X_{16,7} = 0$
8	$X_{8,6} - X_{4,8} - X_{5,8} = 1$
9	$X_{9,29} - X_{4,9} - X_{24,9} - X_{27,9} = 1$
10	$X_{10,31} - X_{28,10} - X_{29,10} = 1$
11	$X_{11,29} - X_{17,11} - X_{14,11} - X_{16,11} = 1$
14	$X_{14,11} + X_{14,16} - X_{19,14} = -1$
15	$X_{15,16} - X_{17,15} = -2$
16	$X_{16,7} + X_{16,11} - X_{14,16} - X_{15,16} = 0$
17	$X_{17,11} + X_{17,15} - X_{36,17} = 0$
19	$X_{19,1} + X_{19,21} + X_{19,14} - X_{20,19} = 0$
20	$X_{20,19} - X_{26,20} - X_{27,20} = -1$
21	$X_{21,22} + X_{21,25} - X_{19,21} = -1$
22	$X_{22,23} + X_{22,26} + X_{22,27} - X_{21,22} = 0$
23	$X_{23,24} + X_{23,31} - X_{22,23} = -1$
24	$X_{24,29} - X_{4,24} - X_{23,24} - X_{26,24} = 0$
25	$X_{25,33} - X_{1,25} - X_{5,25} - X_{21,25} = 1$
26	$X_{26,20} + X_{26,4} + X_{26,24} - X_{22,26} = 0$
27	$X_{27,20} + X_{27,9} - X_{36,27} - X_{22,27} = -1$
28	$X_{28,36} + X_{28,10} + X_{28,32} - X_{29,28} = 0$
29	$X_{29,36} + X_{29,10} + X_{29,28} + X_{29,32} - X_{6,29} - X_{9,29} - X_{11,29} = 0$
31	$X_{31,33} - X_{23,31} - X_{10,31} - X_{32,31} = 1$
32	$X_{32,31} - X_{28,32} - X_{29,32} = 1$
33	$X_{33,35} + X_{33,36} - X_{25,33} - X_{31,33} = -2$
35	$X_{35,36} - X_{33,35} = 2$
36	$X_{36,17} + X_{36,27} - X_{35,36} - X_{33,36} - X_{28,36} - X_{29,36} = 0$

We define

$$A_{ir} = \{i \mid arc\ (r,i)\ exists\}$$

$$B_{jr} = \{j \mid arc\ (j,r)\ exists\}$$

where node r is an eliminated node; note that $X_{j,r} = X_{r,i}$ for all (j, r) and (r, i). This follows, once node r is eliminated, the flow conservation relations are written as

$$|A_{ir}| + \sum_j X_{i,j} = \sum_j X_{j,i} + |B_{jr}| \qquad \forall\, r \in E,\ \forall\, i,j$$

incident to eliminated node r

Using the results of Property 1 and Property 2, the following constraints are added to the model

$$X_{ij} \geq d_i^+ \qquad \forall\, i \mid d_i^- = 1$$

$$X_{ij} \geq d_i^- \qquad \forall\, i \mid d_i^+ = 1$$

We thereafter refer set E as the set of eliminated nodes. Hence, we redefine $N = N / E$.

3.3.2 Finding An Efficient Point

Consider the following modified (P_C) model,

$$Minimize \sum_{(i,j)\in A} c_{ij} \times X_{ij}$$

s.t.

$$X_{ij} \geq 1 \tag{5}$$

$$X_{ij} \geq d_i^+ \qquad \forall\, i \mid d_i^- = 1 \tag{6}$$

$$X_{ij} \geq d_i^- \qquad \forall\, i \mid d_i^+ = 1 \tag{7}$$

$$\sum_j X_{ij} = \sum_j X_{ji} \qquad \forall\, i \in N \tag{8}$$

$$|A_{ir}| + \sum_j X_{i,j} = \sum_j X_{j,i} + |B_{jr}| \qquad \forall\, r \in E, \forall\, i,j$$

incident to eliminated node r (9)

We hereafter refer to constraint sets (5) to (9) as $x \in X$.

Let Z_C^* be the its optimal Z_C value. Z_C^* is a valid lower bound on the Z_C values of all efficient solutions. Any optimal solution to (P_C) may not be efficient since there may exist alternative optimal solutions having smaller Z_D values.

Among the alternative optimal solutions to (P_C), the one having the smallest Z_D value requires the solution of the following problem, which is called (P_D, Z_C^*).

$$Minimize \sum_{(i,j)\in A} d_{ij} \times X_{ij}$$

subject to $x \in X$

$$\sum_{(i,j)\in A} c_{ij} \times X_{ij} = Z_C^*$$

Note that the model above requires solving the (P_C) problem. In place of solving the (P_C) and then the (P_D, Z_C^*) problem, one may solve the following problem:

$$Minimize \left(\sum_{(i,j)\in A} c_{ij} \times X_{ij}\right) + \in_D \times \left(\sum_{(i,j)\in A} d_{ij} \times X_{ij}\right)$$

subject to $x \in X$

for a sufficiently small value of $\in_D$.

The value of $\in_D$ should be small enough that the smallest $\sum_{(i,j)\in A} c_{ij} \times X_{ij}$ value should not increase even for the largest possible value of $\sum_{(i,j)\in A} d_{ij} \times X_{ij}$, which is denoted by D_{max}. Accordingly,

$$Z_C^* + \in_D \times D_{max} \leq Z_C^* + 1 + \in_D \times D_{min} \quad (5)$$

where D_{min} is equal to the smallest possible value of the total distance.

Finding D_{min} and D_{max} Values

The equation (5) follows that

$$\in_D \times (D_{max} - D_{min}) \leq 1$$

$$\epsilon_D \leq \frac{1}{(D_{max} - D_{min})}$$

In our experiments, we first solve (P_C) and (P_D) in order to get D_{max} and D_{min} respectively, and then we set ϵ_D as follows:

$$\epsilon_D = \frac{1}{(D_{max} - D_{min}) + 1}$$

D_{min} is the Z_D value of the P_D problem, i.e.,

$$Minimize \sum_{(i,j)\in A} d_{ij} \times X_{ij}$$

subject to $x \in X$

D_{max} is an upper bound on the Z_D value of all efficient solutions; hence, it is the $(\sum_{(i,j)\in A} d_{ij} \times X_{ij})$ value of the (P_C) problem. In other words, first we solve the (P_C) problem and obtain optimal X_{ij} values, then by using those X_{ij} values in equation of $(\sum_{(i,j)\in A} d_{ij} \times X_{ij})$, we obtain the D_{max} value.

The following example illustrates ϵ_D computations.

Example. Consider the example instance I depicted in Figure 2. By solving the (P_D) model, we found the optimal solution as $Z_D^* = 3755$ which is equal to the D_{min}. Likewise, by solving (P_C) problem, whose optimal solution gives an upper bound on the $(\sum_{(i,j)\in A} d_{ij} \times X_{ij})$ value, we found D_{max} value as 3917.

$$Z_C = \sum_{(i,j)\in A} c_{ij} \times X_{ij}^* = 3700$$

By using the X_{ij} values of the (P_C) problem, we found

$$D_{max} = \sum_{(i,j)\in A} d_{ij} \times X_{ij} = 3917$$

$$
\begin{aligned}
D_{max} \;=\; & 40\times 5+30\times 4+12\times 1+40\times 5+12\times 1+30\times 1+50\times 1 \\
& +50\times 5+15\times 1+50\times 3+9\times 1+50\times 1+30\times 1+60\times 1 \\
& +70\times 1+60\times 5+60\times 1+9\times 1+36\times 3+80\times 1+70\times 1 \\
& +30\times 1+90\times 1+20\times 3+70\times 1+21\times 2+80\times 5+40\times 1 \\
& +20\times 1+60\times 3+20\times 1+40\times 4+30\times 1+30\times 5+40\times 1 \\
& +70\times 1+30\times 5+30\times 1+70\times 5+30\times 1+30\times 1+20\times 1 \\
& +20\times 1
\end{aligned}
$$

As a result, the $\in_D$ value is found as follows:

$$\in_D = \frac{1}{(3917 - 3755) + 1} = 0.00614$$

3.3.3 Finding the Set of All Efficient Points

An optimal solution to the following constrained optimization problem, (P_C, k), is efficient (see, Haimes et al., 1971).

$$Minimize \quad Z_C + \in_D \times Z_D$$

$$\textit{subject to } x \in X$$

$$Z_D \leq k$$

This follows, the efficient set can be generated by solving (P_C, k) for all *k* values in interval $[D_{min,}\, D_{max}]$.

The following procedure systematically varies the value of *k* and returns the set of all efficient points.

Procedure 1 ***(Classical Approach – CA)*** **- Finding the Efficient Set**

Step 0. Calculate the $\in_D$ value which will be used in (P_C, k) model.

- Solve (P_C) model $\quad Minimize \quad Z_C$

  $subject\ to\ x \in X$

Let $D_{\max}$ be the Z_D value of the P_C

- Solve (P_D) model $\quad Minimize \quad Z_D$

  $subject\ to\ x \in X$

Let $D_{\min}$ be the Z_D value of the P_D

Let

$$\in_D = \frac{1}{(D_{max} - D_{min}) + 1}$$

Step 1. Solve

$$Minimize \quad Z_C + \in_D \times Z_D$$

$$subject\ to\ x \in X$$

$$Z_D \leq k$$

Let (Z_C^*, Z_D^*) be the solution.

Set $r = r + 1$

Step 2. If $Z_D^* = D_{\min}$, then go to Step 3. Otherwise, set $k = Z_D^* - 1$ and go to Step 1.

Step 3. Stop, if all r efficient points are generated.

Note that each iteration of the algorithm generates an efficient point. When the procedure terminates at Step 5, whole set of r efficient points are generated.

The number of efficient points, r, takes value of at most $(D_{max} - D_{min}) + 1$. Hence, the algorithm iterates pseudo polynomial number of times.

3.3.4 An Example

We illustrate Procedure 1 via our example instance I. The stepwise execution is as stated below:

Step 0. Solve Z_C

subject to $x \in X$

The optimal solution gives a D_{max} value of 3917. We set *k* value to 3917.

Solve Z_D

subject to $x \in X$

The optimal solution gives a D_{min} value of 3755.

$$\in_D = \frac{1}{(3917 - 3755) + 1} = 0.00614$$

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3917$

The optimal solution is $(Z_C^*, Z_D^*) = (3700, 3917)$.

r value is set to 1.

Step 2. $Z_D > 3755$, *k* = 3916

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3916$

The optimal solution is $(Z_C^*, Z_D^*) = (3730, 3891)$.

r value is set to 2.

Step 2. $Z_D > 3755$, *k* = 3890

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3890$

The optimal solution is $(Z_C^*, Z_D^*) = (3760, 3865)$.

r value is set to 3.

Step 2. $Z_D > 3755$, *k* = 3864

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3864$

The optimal solution is $(Z_C^*, Z_D^*) = (3800, 3845)$.

r value is set to 4.

Step 2. $Z_D > 3755$, *k* = 3844

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3844$

The optimal solution is $(Z_C^*, Z_D^*) = (3840, 3825)$.

r value is set to 5.

Step 2. $Z_D > 3755$, *k* = 3824

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3824$

The optimal solution is $(Z_C^*, Z_D^*) = (3890, 3819)$.

r value is set to 6.

Step 2. $Z_D > 3755$, *k* = 3818

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3818$

The optimal solution is $(Z_C^*, Z_D^*) = (3940, 3813)$.

r value is set to 7.

Step 2. $Z_D > 3755$, *k* = 3812

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3812$

The optimal solution is $(\mathrm{Z_C^*}, \mathrm{Z_D^*}) = (3990, 3807)$.

r value is set to 8.

Step 2. $Z_D > 3755$, *k* = 3806

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3806$

The optimal solution is $(Z_C^*, Z_D^*) = (4090, 3801)$.

r value is set to 9.

Step 2. $Z_D > 3755$, *k* = 3800

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3800$

The optimal solution is $(Z_C^*, Z_D^*) = (4189, 3795)$.

r value is set to 10.

Step 2. $Z_D > 3755$, *k* = 3794

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3794$

The optimal solution is $(Z_C^*, Z_D^*) = (4400, 3785)$.

r value is set to 11.

Step 2. $Z_D > 3755$, *k* = 3784

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3784$

The optimal solution is $(Z_C^*, Z_D^*) = (4610, 3775)$.

r value is set to 12.

Step 2. $Z_D > 3755$, *k* = 3774

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3774$

The optimal solution is $(Z_C^*, Z_D^*) = (4820, 3765)$.

r value is set to 13.

Step 2. $Z_D > 3755$, *k* = 3764

Step 1. Solve $Z_C + 0.00614\, Z_D$

subject to $x \in X$

$Z_D \leq 3764$

The optimal solution is $(Z_C^*, Z_D^*) = (5030, 3755)$.

r value is set to 14.

Step 2. Since $Z_D = 3755$, we go to step 3.

Step 3. 14 efficient points are generated, then we stop the algorithm.

The set of generated efficient points is given in Table 5 below.

Table 5: Set of Efficient Points for our Example Instance I

(N, M) = (25, 43)			
r	**k**	$\mathbf{Z_C^*}$	$\mathbf{Z_D^*}$
1	3917	3700	3917
2	3916	3730	3891
3	3890	3760	3865
4	3864	3800	3845
5	3844	3840	3825
6	3824	3890	3819
7	3818	3940	3813
8	3812	3990	3807
9	3806	4090	3801
10	3800	4189	3795
11	3794	4400	3785
12	3784	4610	3775
13	3774	4820	3765
14	3764	5030	3755

The set of generated efficient points is also represented in Figure 3 below:

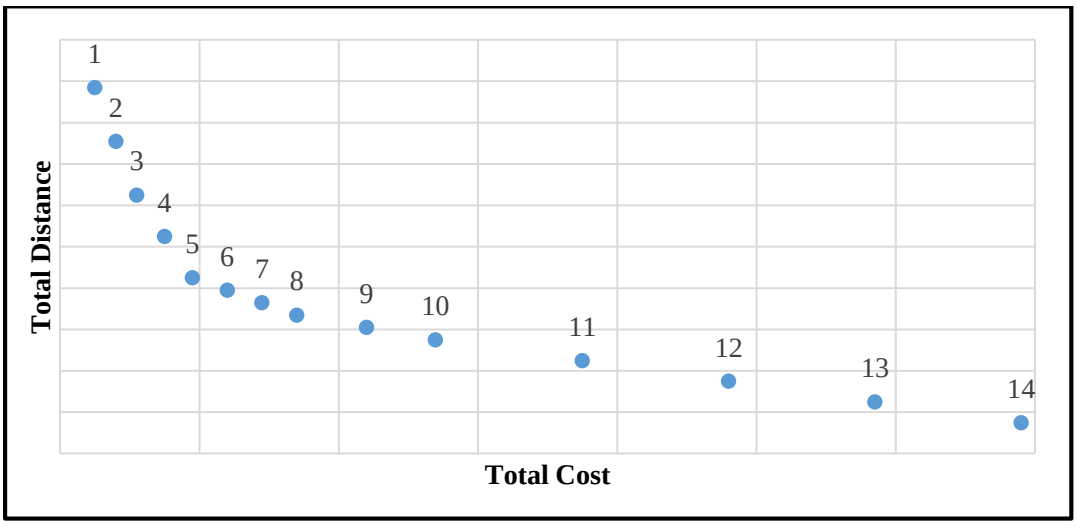


Figure 3: The Efficient Points' Graph

CHAPTER 4

THE COMPLEXITY AND BRANCH & BOUND ALGORITHM

In this section, we first settle the complexity of the constrained CPP, i.e., $P_{c,}k$, and then present a branch and bound algorithm for its solution.

Theorem 1 settles the complexity of the problem.

Theorem 1. The constrained CPP is strongly NP-hard.

Proof. Recall that the CPP reduces to the transportation problem (see, page 13). Hence, our constrained (capacitated) CPP problem is equivalent to a transportation problem with the following additional constraint.

$$\sum_{i \in I} \sum_{j \in J} d_{ij} \times X_{ij} \leq \overline{D}$$

where $\overline{D} = k - \sum_{(i,j) \in A} d_{ij}$; hence it is a constant (playing a role of updated capacity).

When $s_i = d_j = 1 \quad \forall\, i, j,$ the capacitated transportation problem reduces to the capacitated, so called, Generalized Assignment Problem (GAP).

It follows that our constrained CPP reduces to the GAP. The GAP is NP-hard in the strong sense (see, Martello and Toth (1990)), so is our constrained CPP.

□

The problem of generating a single efficient point is strongly NP-hard as the constrained CPP is strongly NP-hard. It follows that the problem of generating all

efficient solutions with respect to our total cost and total distance criteria is strongly NP-hard.

Attributing to the complexity of the problem, our procedure (Procedure 1) to generate all efficient points is likely to fall into computational burden. To dispel the burden to some extent, we present an implicit enumeration technique – a branch and bound algorithm. Our aim is to attain optimal solutions to large sized instances in reasonable solution times.

Our branch and bound algorithm generates one efficient point at a time. It starts with an initial upper bound (incumbent solution) and updates it whenever a complete solution with a better objective function value is reached.

4.1 Branch and Bound Algorithm

4.1.1 Initial Upper Bound

We solve the Linear Programming Relaxation (LPR), i.e., first relax the integrality constraints on the X_{ij} values and then solve the resulting problem to optimality.

We benefit from the optimal solution of the LPR to find an initial upper bound on the Z_c value. Below is the stepwise description of our upper bounding procedure.

Procedure 2 - Finding an Initial Upper Bound

Step 1. Solve the LPR.

Step 2. Let cτ be a cycle that resides all fractional variables. Let X_{rs}^{L} be the smallest fractional variable in $c\tau$.

Step 3. Let $X_{rs} = \lceil X_{rs}^{L} \rceil$ and update the fractional variables around $c\tau$ so as to preserve the flow conservation constraints.

Step 4. If $X_{rs} = \lceil X_{rs}^{L} \rceil$ gives feasible solution, then go to Step 5.

Else let $X_{rs} = \lfloor X_{rs}^{L} \rfloor$ and update the fractional variables around cτ.

Step 5. If all variables are integers, stop.

Else go to Step 2.

We illustrate our upper bound via our example instance I.

Example. The optimal LPR solution of the (P_C, k) problem with k value of 3794 as stated below:

$X^*_{1,2} = 5,\ X^*_{2,4} = 4,\ X^*_{4,3} = 4,\ X^*_{5,1} = 5,\ X^*_{6,5} = 3,\ X^*_{7,6} = 3,\ X^*_{3,9} = 5,$
$X^*_{9,10} = 5,\ X^*_{12,11} = 3,\ X^*_{15,12} = 3,\ X^*_{16,15} = 3,\ X^*_{10,13} = 3,\ X^*_{13,14} = 4,$
$X^*_{17,16} = 4,\ X^*_{18,17} = 5,\ X^*_{20,18} = 5,\ X^*_{21,20} = 4.9,\ X^*_{14,21} = 4.9,\ X^*_{22,23} = 1.1,$
$X^*_{14,22} = 1.1, X^*_{23,24} = 1.1, X^*_{24,25} = 1.1, X^*_{25,20} = 1.1$

and all other X^*_{ij} values are equal to 1.

The optimal total cost, $Z_{C,k}$, is equal to:

$$Z_{C,k} = \sum_{(i,j)\in A} c_{ij} \times X^*_{ij}$$

$$\begin{aligned} Z_{C,k} = {} & 40 \times 5 + 30 \times 1 + 40 \times 4 + 40 \times 5 + 40 \times 4 + 30 \times 1 + 50 \times 1 + 50 \times 5 \\ & + 50 \times 3 + 50 \times 1 + 30 \times 3 + 50 \times 1 + 30 \times 1 + 60 \times 1 + 70 \times 1 \\ & + 60 \times 5 + 60 \times 1 + 60 \times 3 + 30 \times 1 + 80 \times 1 + 70 \times 1 + 30 \times 1 \\ & + 90 \times 1 + 20 \times 3 + 70 \times 1 + 70 \times 4 + 20 \times 4.9 + 40 \times 1.1 + 40 \\ & \times 3 + 40 \times 1 + 40 \times 3 + 40 \times 4 + 30 \times 1 + 30 \times 5 + 40 \times 1 + 70 \\ & \times 4.9 + 30 \times 5 + 30 \times 1 + 20 \times 1 + 30 \times 1.1 + 30 \times 1.1 + 80 \times 1.1 \\ & + 70 \times 1.1 \end{aligned}$$

$$Z_{C,k} = 4211$$

According to the solution of the LPR, the following cycle resides all fractional variables:

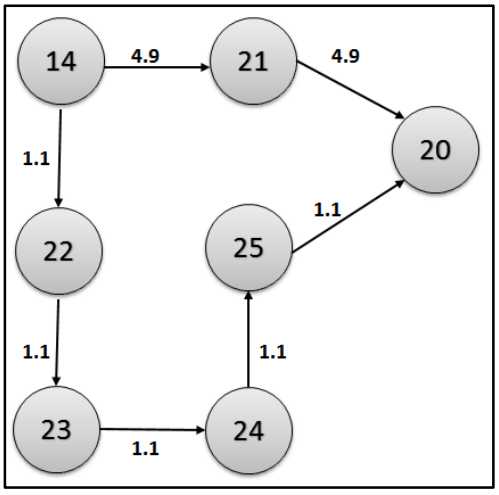


Figure 4: Cycle with Fractional Variables

We choose one of the smallest fractional variables in the cycle, i.e. $X_{rs}^{L} = X_{14,22} = 1.1$.

Then, we let $X_{14,22} = 2$ and update the fractional variables around the cycle as follows:

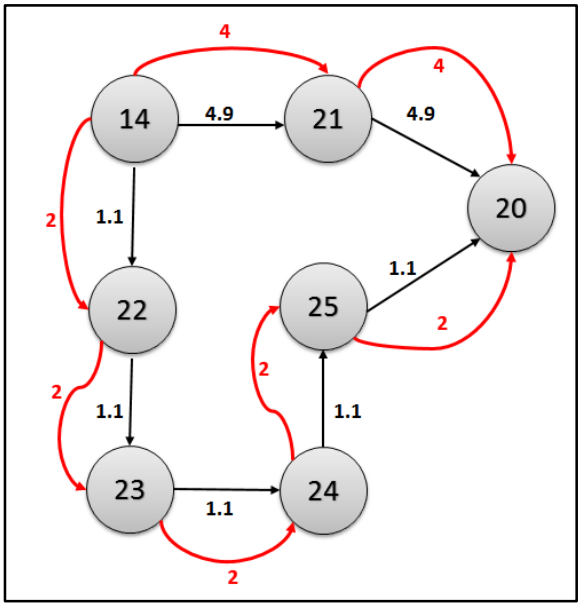


Figure 5: Cycle with Updated Variables

Initial upper bound in terms of total cost is then equal to:

$$UB_C^0 = 4211 + 0.9\,(40 + 30 + 30 + 80 + 70 - 20 - 20) = 4400$$

Since all variables are integers, we find the upper bound value and stop the algorithm.

4.1.2 Branching Scheme

At each node of the branch and bound tree, we solve the LPR and explore the tree using the fractional variables of the LPR solution.

We let X_{rs} be the maximum value among the fractional variable set and generate the following two child nodes:

Child Node I. Add constraint $X_{rs} \geq \lceil X_{rs}^L \rceil$

Child Node II. Add constraint $X_{rs} \leq \lfloor X_{rs}^L \rfloor$

At each node, we solve the LPR and use its optimal solution value as a lower bound. We select the node with smaller lower bound. For the selected node, we find an upper bound using the upper bounding procedure.

The following figure illustrates our branch and bound tree.

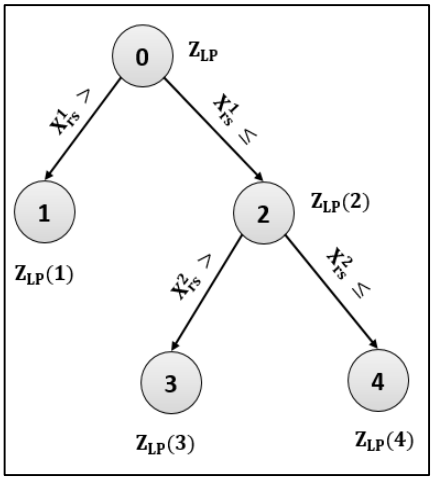


Figure 6: BAB Tree

According to the tree, the LPR is solved first and two nodes 1 and 2 are generated. Node 2 is selected for further branching as $Z_{LP}^{2} \leq Z_{LP}^{1}$ and from Node 2 two further nodes 3 and 4 are created. Node 3 and Node 4 use LP relaxations of Node 2 with additional constraints of $X_{rs}^{2} \geq \lceil X_{rs}^{2} \rceil$ and $X_{rs}^{2} \leq \lfloor X_{rs}^{2} \rfloor$, respectively. The tree will further explore from Node 3 if $Z_{LP}^{3} \leq Z_{LP}^{4}$ and from Node 4 if $Z_{LP}^{3} > Z_{LP}^{4}$.

A node is fathomed if one of the following conditions hold:

i. The resulting LPR leads to an infeasible solution.
ii. The resulting LPR gives all integer decision variables. In such a case, the solution (Z_C^I, Z_D^I) is updated if its $Z_C < Z_C^I$ or $Z_C = Z_C^I$ and $Z_D < Z_D^I$.
iii. The resulting LPR's objective function value is no better than the incumbent solution. That is, $Z_C^{LP} > Z_C^I$ or $(Z_C^{LP} = Z_C^I$ and $Z_D^{LP} > Z_D^I)$.

We employ a depth first strategy due to its relatively low memory requirements. According to this strategy, we start form the root node and explore branching from the node having smaller Z_C value (or smaller Z_D value if the Z_C values are equal).

If both nodes are fathomed, we backtrack to the previous level. We stop when we backtrack to Level 1. The best solution at termination, i.e., incumbent solution is the optimal solution.

We illustrate our branching scheme on example instance I.

Example. Recall that for instance I, the optimal solution of the LPR at the initial level equals to 4211 when k = 3794. The initial upper bound is found as 4400.

The BAB tree of the instance is presented in Figure 7. The figures on the nodes indicate the order at which it is created. At each node, the lower and upper bound values are also reported.

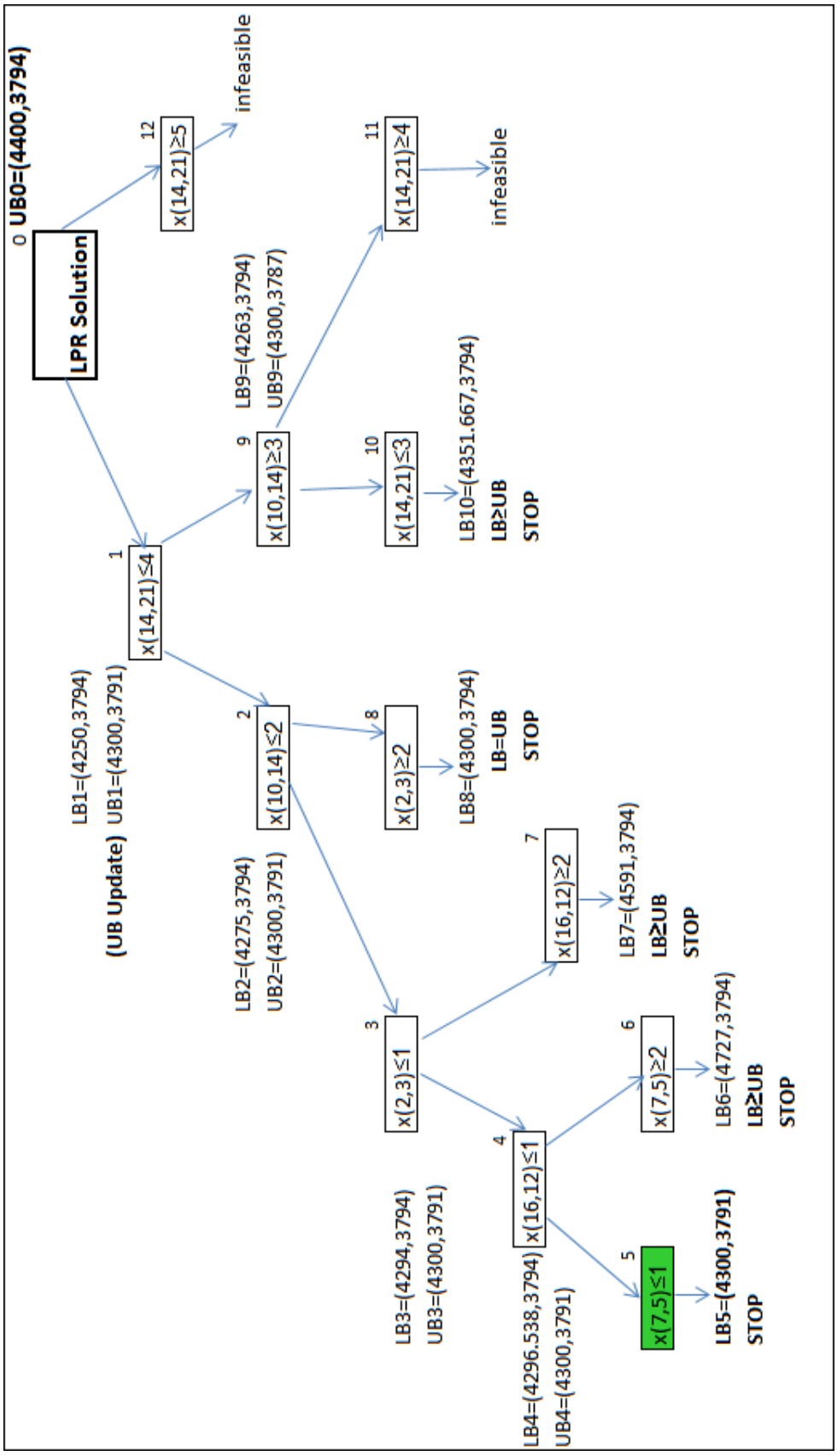


Figure 7: BAB Tree of the Instance

At the Node 1, the following LPR is solved:

$$Minimize\ Z_C + \in_D \times Z_D$$

$$Subject\ to \quad x \in X$$

$$X_{14,21} \leq 4$$

The optimal solution of associated LPR is found as $Z_{C,k} = 4250$ for the value of k = 3794. We also update the upper bound in here.

According to the solution of LPR, the cycle that resides fractional variables is given in Figure 8 below. We choose the smallest fractional variable in the cycle, i.e., $X_{10,13}$ = 1.5. Then, we let $X_{10,13} = 2$ and update the fractional variables around the cycle as shown in the figure below.

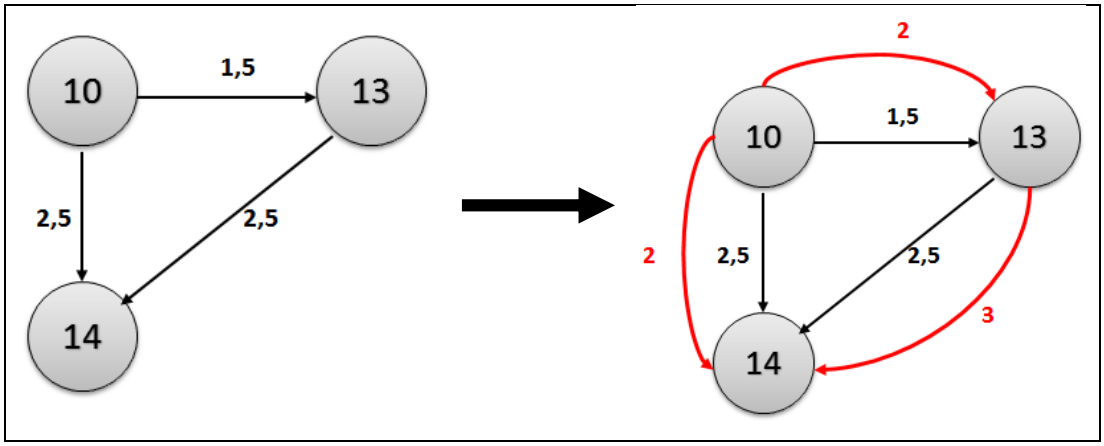


Figure 8: Upper Bounding Procedure

The upper bound in terms of total cost and total distance is then equal to:

$$UB_C^1 = 4250 + 0.5\,(60 - 30 + 70) = 4300$$

$$UB_D^1 = 3794 + 0.5\,(9 - 36 + 21) = 3791$$

Then, we update the upper bound from $(UB_C, UB_D) = (4400, 3794)$ to $(4300, 3791)$.

At the Node 2, we solve the following LPR model:

$$Minimize\ Z_C + \in_D \times Z_D$$

$$subject\ to \quad x \in X$$

$$X_{14,21} \leq 4,\ X_{10,14} \leq 2$$

The optimal solution of the above LPR is found as $Z_{C,k} = 4275$ for the value of k = 3794. The upper bound found at this node is no better than the one found in Node 1. Therefore, it is not updated.

According to the solution of the LPR at Node 2, the cycle that resides fractional variables are $X_{2,3} = 1.5,\ X_{2,4} = 3.5,\ X_{4,3} = 3.5$.We choose the smallest fractional variable in the cycle, i.e., $X_{2,3} = 1.5$. Then, at the Node 3, we solve the following LPR model:

$$Minimize\ Z_C + \in_D \times Z_D$$

$$Subject\ to \quad x \in X$$

$$X_{14,21} \leq 4,\ X_{10,14} \leq 2,\ X_{2,3} \leq 1$$

The optimal solution of the above LPR is found as $Z_{C,k} = 4294$ for the value of k = 3794.

According to the solution of LPR at Node 3, now $X_{15,12} = 2.85,\ X_{16,12} = 1.15,$ $X_{16,15} = 2.85$ resides fractional cycle. We choose the smallest fractional variable in the cycle, i.e., $X_{16,12} = 1.15$. Then, at the Node 4, we solve the following LPR model:

$$Minimize\ Z_C + \in_D \times Z_D$$

$$Subject\ to \quad x \in X$$

$$X_{14,21} \leq 4,\ X_{10,14} \leq 2,\ X_{2,3} \leq 1,\ X_{16,12} \leq 1$$

The optimal solution of the LPR is found as $Z_{C,k} = 4296{,}538$ for the value of k = 3794. According to the solution of LPR at Node 4, the cycle that resides fractional variables are now $X_{6,5} = 2.885,\ X_{7,5} = 1.115,\ X_{7,6} = 2.885.$

At Child Node 5, when we solve the following LPR by choosing the smallest fractional variable in the cycle, the optimal solution is equal to the updated upper bound value at Node 1, i.e., $(LB_C, LB_D) = (4300, 3791)$. Since the resulting LPR gives all integer decision variables, we stop branching from Node 5 and continue to explore from the other node of the same level.

$$Minimize\ Z_C + \in_D \times Z_D$$

$$Subject\ to \quad x \in X$$

$$X_{14,21} \leq 4,\ X_{10,14} \leq 2,\ X_{2,3} \leq 1,\ X_{16,12} \leq 1,\ X_{7,5} \leq 1$$

Nodes 6, 7, 8 and 10 are fathomed since the objective functions of the resulting LPRs are greater than or equal to those of the incumbent solution.

Nodes 11 and 12 are also fathomed as their since the resulting LPRs leads to an infeasible solution.

The best solution at termination is $(4300, 3791)$ is optimal and it is found at Node 5.

CHAPTER 5

COMPUTATIONAL EXPERIMENTS

In this section, we first report on our data generation scheme and performance measures. We then discuss the results of our preliminary experiments and main experiment.

5.1 Data Generation

The cost and distance weights associated with each arc are generated from the discrete uniform distribution between 1 and 100. In order to obtain random arc weights, we first use Combined MRG (Multiple Recursive Generators) technique, which is proposed by L'Ecuyer (1999), to generate uniformly distributed numbers between 0 and 1, denoted by $U \sim U(0,1)$. We then use Inverse Transform (IT) method to turn uniformly distributed numbers between 0 and 1 into discrete Uniform distribution between 1 and 100, denoted by $X \sim DU(1,100)$, using $X = 1 + \lfloor (100 - 1 + 1)\, U \rfloor$.

The computational results are based on 7 different networks. In order to observe the effect of the number of nodes on the problem difficulties, we use 3 different problem sizes. For each size, we use different number of arcs to see the effect of the number of arcs on the problem difficulties. The number of nodes and arcs used in our experiments are shown in Table 6 below.

Table 6: All N and M values used in Computational Experiments

Number of Nodes (N)	100	100	300	300	300	500	500
Number of Arcs (M)	200	400	600	675	750	1200	1300

The network with (N, M) values of (100, 200) is taken from the official website of University of Brescia, Department of Information Engineering (http://www.ing.unibs.it/~orgroup/instances.html). In order to make the network strongly connected, we slightly modify the instance by randomly adding a few arcs. The network with 100 nodes and 400 arcs, on the other hand, is obtained by combining the arcs of two different instances with 100 nodes and 200 arcs.

The networks with (N, M) values of (500, 1200) and (500, 1300) are taken from the webpage, "Arc Routing Problems: Data Instances" maintained by Corberán A. et al (http://www.uv.es/corberan/instancias.htm). The networks with 300 nodes are obtained by modifying another instance taken from the webpage.

In our preliminary experiments, we use the networks with (N, M) values of (100, 200), (300, 600), (300, 675), (500, 1200) and (500, 1300). In our main experiment, we use all 7 combinations reported in Table 6.

All experiments are carried out on an Intel(R) Core(TM) i5-3317U and clocked at 1.70 GHz with 4 GB RAM. In order to generate whole efficient set, optimization by software algorithm is coded in C++ (CodeLite version 6.0) where MIP are solved consecutively by using GAMS Cplex Solver. The branch and bound algorithm, on the other hand, is coded in Java (Eclipse Luna version 4.4.0).

5.2 Performance Measures

For the branch and bound algorithm, we report the solution times in seconds and the number of partial solutions (nodes), that is the number of nodes travelled in a branch and bound tree. The solution times are expressed in central processing unit (CPU) in seconds. For both CPU times and number of nodes, we give the results in terms of the average and maximum values over 10 problem instances.

For the Classical Approach – CA (see, page 27), we evaluate the performance with average and maximum solution times in terms of CPU in seconds over 10 instances.

We also report the average and the maximum number of efficient points over 10 problem instances for all networks with (N, M) pairs.

We set an upper limit of 3600 seconds for the execution of the branch and bound algorithm and classical approach.

5.3 Preliminary Experiments

In our preliminary runs, we first investigate the effects of the correlation between two arc weight sets on the difficulty of the problems. In order to see the effect of correlation, we generate three different set of arc weights which are stated as random set, negatively correlated set and positively correlated set.

Random set is obtained by generating randomly all arc weights, which are denoted by c_{ij} and $d_{ij,}$ from discrete uniform distribution between 1 and 100.

To generate the negatively correlated set, we ordered c_{ij} and d_{ij} values in such a way that $c_{rs} \leq c_{uv}$ implies $d_{rs} \geq d_{uv}$ for all (c_{rs}, d_{rs}) and (c_{uv}, d_{uv}) combinations where (rs) and (uv) combinations range from 1 to N.

Finally, in order to generate positively correlated random set, we conserved the first half of the arc weights in the random. Then for the second half of the data, which corresponds to the weights of arcs in the second half, we modified the c_{ij} and d_{ij} values in such a way that $c_{rs} = d_{rs}$ for all (rs) combinations.

To observe the effect of correlation between arc weight sets, we select three networks with (N, M) values of (100, 200), (300, 600) and (300, 750). We report the average and maximum number of efficient points for each data set in Table 7 below.

Table 7: The Effect of Correlation on # of the Efficient Point

		Positive		**Random**		**Negative**	
N	**M**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
100	200	16	35	43	150	12	14
300	600	53	102	106	152	284	436
300	675	166	413	223	421	875	1162

As can be observed from the table above, the number of efficient points is very sensitive to the correlation between the arc weight sets.

When there is a positive correlation between arc weights, the optimal solution of single objective Chinese Postman Problem c_{ij} values is very close to that of with d_{ij} values. Hence we do have considerably small number of efficient points. Note that, the average and maximum number of efficient points are not too sensitive to the increases in number of nodes and arcs.

When c_{ij} and d_{ij} values are negatively correlated, the best solution for the single objective problem with c_{ij} values is likely to be far from being good for the single objective problem with d_{ij} values. Since our two objectives are expectedly conflicting, the number of efficient points per instance is very high. Therefore, the conflicting nature of the problem strongly affects the number of efficient points. For example, as it is seen from Table 7, while the network with (N, M) value of (100, 200) has average number of 12 efficient points, the network with 300 nodes and 675 arcs has 875 efficient points on average. That is, by enlarging the size of the network only 3 times, the average number of efficient point becomes approximately 73 times larger.

For the random data set, in terms of average and maximum number of solutions, we have fewer solutions than negatively correlated data set, while we have much more solutions than positively correlated data for the networks with (N, M) value of (300, 600) and (300, 675). We observe inconsistent behavior of the efficient point sets. This can be observed from the gap between the maximum and average number of efficient points. This gap is higher than that of between negatively correlated and positively correlated cases when we consider the networks with (N, M) value of (100, 200) and (300, 675). Note that when the network has 100 nodes and 200 arcs, the gap between the average and the maximum is 107 for the random data set, while it is only 19 and 2 for the positively and negatively correlated data sets, respectively.

In Table 8 below, we report on the performance of the branch and bound algorithm per efficient point basis.

Table 8: The Effect of Correlation on the Performance of the BAB-per eff. point in seconds

			Positive		Random		Negative	
N	**M**		**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
100	200	**CPU Time (sec.)**	0.56	1.81	0.382	1.069	0.44	0.49
		Nodes	9	18	16	27	5	6
300	600	**CPU Time (sec.)**	0.71	0.98	1.01	1.319	1.87	2.38
		Nodes	35	62	70	96	149	189
300	675	**CPU Time (sec.)**	1.12	1.7	1.724	2.485	4.09	4.78
		Nodes	73	123	121	173	297	352

We observe from Table 8 that an efficient point is easiest to obtain when there is a positive correlation. This is due to the fact that the knapsack constraint used in generating the efficient points becomes almost redundant; therefore, the problem becomes close to a single objective unconstrained problem, which can be solved in polynomial time. Note that, for the positively correlated data set, increasing the size of the network does not strongly affect average and maximum CPU to find an efficient point.

Note also from Table 8 that the hardest problem instances are from the negatively correlated data set. The reason is that, for those instances, the knapsack constraint ($\sum d_{ij}X_{ij} \leq D$) becomes binding, and this binding constraint leads to an increase in the complexity of the problem significantly. As it is seen from Table 8 for the negatively correlated data set, an efficient point can be found only in 0.44 seconds on average for the network with 100 nodes and 200 arcs, while it takes 4.09 seconds on average to find an efficient point for the network with (N, M) values of (300, 675) which is approximately 8 times larger.

We also observe that as the problem size increases, the gap between the CPU times of the negatively and positively correlated data set also increases. For the network with (N, M) values of (300, 675), it takes 1.12 seconds on average to find an efficient point for the positively correlated set, while it reaches to 4.09 seconds for the same problem combination. Note from Table 8 that, the average CPU time of the network with N=100 and M=200 for the positively correlated set is greater than that of the negatively correlated set. This inconsistent behavior is due to an instance whose solution time to find all efficient points is 1.81 seconds.

Table 8 also shows that the performance of the branch and bound algorithm is least consistent for the random data set. This can be observed from the high gap between the maximum and average CPU times. Note that the gap between the average and maximum CPU of the network with 300 nodes and 675 arcs for the random set is noticeably greater than those of the negatively and positively correlated data sets. It is likely that when the instances are closer to negatively correlated set, the CPU times are higher, when they are closer to positively correlated set, on the other hand, the CPU times are lower.

If we consider the number of nodes generated during the branch and bound algorithm, for all data sets, it is seen from the Table 8 that as the problem size increases, the average and maximum number of nodes also increase expectedly. We can observe the effect of problem size expansion most remarkably in the negatively correlated data set. Note that for the network with values of N=100 and M=200, 5 nodes were generated on average, whereas for the N=300 and M=675, the algorithm generated 297 nodes on average. For the positively correlated data set, on the other hand, for the same problem combinations, the average number of nodes increases from 9 to 73. For the network with (N, M) values of (100, 200), although the fewest number of average and maximum nodes appear in the negatively correlated case, it is seen that average CPU time is still higher than that of the random data set which might be due to the higher complexity of the problem with negatively correlated data.

In Table 9, we report the same statistics with those of Table 8, but on per instance basis.

Table 9: The Effect of Correlation on the Performance of the BAB-per instance in seconds

			Positive		Random		Negative	
N	**M**		**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
100	200	**CPU Time (sec.)**	5.74	9.78	8.543	19.097	5.26	5.44
		Nodes	265	850	848	3138	63	70
300	600	**CPU Time (sec.)**	37.3	92.12	107.245	182.76	546.51	1020.21
		Nodes	2048	6338	7505	13478	43627	82424
300	675	**CPU Time (sec.)**	190.01	379.59	416.311	1046.33	3560.95	4776.74
		Nodes	12864	26944	29202	72904	259041	347870

We observe the highest number of efficient points and highest CPU times per efficient point for negatively correlated data set. This follows that, the CPU times per instance is highest for the negatively correlated set. As it seen from the table above, in terms of the average and maximum CPU times, the highest solution times correspond to the negatively correlated set for the networks with (N, M) values of (300, 600) and (300, 675). Not only the high CPU time per efficient point, but also too many efficient points per instance, leads to a CPU time of 3560.95 seconds on average which is very close to our termination limit. On the contrary, the combination with N=100 and M=200 shows inconsistent behavior with respect to other networks such that the random data set gives the highest CPU times per instance.

En route to aiming at a random experiment, we hereafter use the random set in our main experiment.

We next analyze the performance of our Shortest Path (SP) rule in reducing the size of the problem. To do so, three problem combinations with N = 100, M = 200; N = 500, M = 1200 and N = 500 and M = 1300 are selected. We report the results in Table 10 below.

Table 10: The Effectiveness of the SP rule

		Eff. Sol.		Reduced Problem Size			
N	**M**	**Avg**	**Max**	**Avg N**	**Max N**	**Avg M**	**Max M**
100	200	43	150	95	98	182	192
500	1200	162	294	484	486	1118	1127
500	1300	355	503	495	497	1279	1287

As can be observed from the table above, the SP rule slightly reduces the number of nodes and the number of arcs. On average, there are 95 reduced nodes for the network with 100 nodes, 486 and 495 reduced nodes for the other networks with 500 nodes. We eliminate 18, 82 and 21 arcs on average over the 200, 1200 and 1300 arcs, respectively.

We observe that for each eliminated node, we remove approximately 4 arcs from the network on average for all three problem combinations.

Any reduction in the number of arcs and number of nodes is important as these reductions directly influence the complexity of the problem. Recall that, as the number of arcs decreases, the number of integer variables, thereby the complexity of the problem also decreases.

We now discuss the effect of the upper bounds used at every node of the branch and bound algorithm. In order the see the effect, we design two algorithms: one with upper bounds and one without upper bounds. In latter algorithm, the upper bounds are updated only when a complete solution is reached.

Table 11 reports the CPU times and number of nodes for both algorithms.

Table 11: The Effect of Upper Bounds-per Eff. Points

		BAB with UB				BAB w/o UB			
		# of Nodes		CPU Time		# of Nodes		CPU Time	
N	**M**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
100	200	16	27	0.382	1.069	35	77	0.427	1.172
300	600	70	96	1.01	1.319	236	323	1.492	2.014
300	675	121	173	1.782	2.51	553	988	3.8	6.758

Table 11 reveals that the upper bounds have significant effect on the performance of the branch and bound algorithm. The number of nodes generated and CPU times decrease drastically once the upper bounds are incorporated. Note that for N = 300 and M = 600 combination, using upper bounds reduces the average number of nodes from 236 to 70 and the maximum number of nodes from 323 to 96. On the other hand, when upper bounds are used, the CPU time reduces to 1.01 seconds from 1.492 seconds and the maximum CPU time reduces to 1.319 seconds from 2.014 seconds. The amount of reduction in terms of the number of nodes is more significant than CPU times as the upper bound calculations require extra time. For instance, for the network with 100 nodes, the average number of nodes reduces from 35 to 16 by upper bounds while CPU time per efficient point reduces very slightly due to the time spent to calculate upper bounds. Our conclusion is that, the time spent to calculate the upper bounds is outweighed by the amount of reduction in the partial solutions.

We also observe that the elimination power of the upper bounds becomes more predominated as the problem size increases. The network with (N, M) values of (300, 675) clearly supports this since by using the branch and bound algorithm with upper bound, the CPU time per efficient point reduces by 2 seconds on average and approximately 4 seconds in maximum time. Likewise, the average and maximum number of nodes decrease approximately 80% and 83% respectively.

In Table 12, we report the same statistics with those of Table 11, but on a per instance basis.

Table 12: The Effect of Upper Bounds-Instance

		BAB with UB				BAB w/o UB			
		# of Nodes		CPU Time		# of Nodes		CPU Time	
N	M	Avg	Max	Avg	Max	Avg	Max	Avg	Max
100	200	848	3138	8.543	19.097	2140	7248	10.616	23.88
300	600	7505	13478	107.245	182.76	25935	40242	159.059	239.88
300	675	29202	72904	426.947	1056.72	139924	415900	956.118	2845.3

From Tables 11 and 12, we observe the fewest number of nodes and smallest CPU times per efficient point, and per instance, for the branch and bound algorithm with upper bound. Note that, for the network with 675 arcs, the average CPU time decreases from 956.118 seconds to 426.947 seconds and the maximum CPU time decreases from 2845.3 seconds to 1056.72 seconds by the upper bounds.

Based on the results of Table 11 and Table 12, we decide to conduct our main experiment with the upper bounds for the branch and bound algorithm.

5.4 Main Experiment

We perform the main experiment with 100, 300 and 500 nodes. For each N value (number of nodes), we try two values for M (number of arcs). See Table 6 for the problem sizes used in our experiment.

For each (N, M) problem combination, we generate 10 problem instances by using different arc weights (c_{ij} and d_{ij} values) which are generated randomly between 1 and 100. Hence, our main experiment set resides 60 problem instances in total. For each instance, we generate the set of all efficient points; hence, we solve many combinatorial optimization problems.

The results related to the number of efficient points are reported in Table 13 below. The table includes the average and maximum number of efficient points for each problem combination.

Table 13: The Number of Efficient Points

		# of Efficient points	
N	**M**	**Average**	**Maximum**
100	200	43	150
100	400	237	425
300	600	106	152
300	750	426	904
500	1200	162	294
500	1300	355	503

The average number of efficient points for each combination is also represented in Figure 9. The figure shows the effect of problem size on the number of efficient points.

As can be observed from Figure 9, the number of efficient points is highly dependent on the problem size. Note from the figure that for N = 100, the average number of efficient points increases from 43 to 237 as M increases from 200 to 400. For N = 300, these increases are from 106 to 426 when M increases from 600 to 750. For N = 500, the average CPU times increases more than two times when M increases by 100.

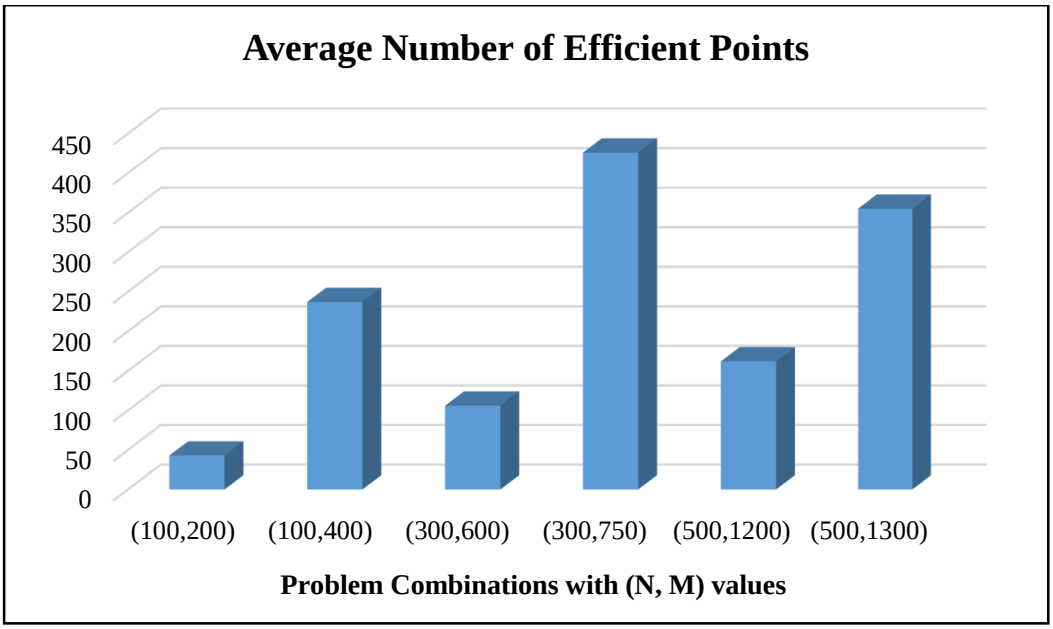


Figure 9: The Average Number of Efficient Points per Problem Combination

Table 14 below reports the performance of the branch and bound (BAB) algorithm in generating an efficient point. The performance is evaluated in terms of the average and maximum CPU times (seconds) and the number of nodes generated.

Table 14: The Performance of BAB and CA-per efficient point in seconds

		CA CPU Time (sec.)		BAB CPU Time (sec.)		BAB Nodes	
N	**M**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
100	200	0.298	0.490	0.382	1.069	16	27
100	400	0.287	0.311	1.243	1.613	141	197
300	600	2.593	3.115	1.01	1.319	70	96
300	750	2.292	2.993	2.779	3.765	177	235
500	1200	5.183	5.727	1.854	2.431	187	384
500	1300	5.286	5.696	4.224	5.563	152	202

We observe from the Table 14 that as the number of nodes (N) or number of arcs (M) increases, the average and maximum solution time to find an efficient point increases significantly. Finding an efficient point takes 0.382 seconds on average for the networks with 100 nodes and 200 arcs, while it reaches to 1.854 seconds for the network with (N, M) values of (500, 1200). This significant increase can be attributed to the increases in the dimensions of the LP models used in finding the lower and upper bounds.

Note from the Table 14 that when N = 100 and M = 200, the average number of nodes generated is 16 and the average CPU time to find an efficient point is 0.382 seconds. For N = 100, when M increases to 400, the average number of nodes and CPU time rise to 141 and 1.243 seconds, respectively. Similarly, when N = 500 nodes, as we increase the number of arcs to 1300, the solution time to find an efficient point increases about 2.37 seconds. Therefore, by increasing the number of arcs, we spend much more effort to find an efficient point.

We report on the performance statistics for generating the efficient set in Table 15.

Table 15: The Performance of BAB and CA-per Instance in seconds

		CA CPU Time (sec.)		**BAB CPU Time (sec.)**		**BAB Nodes**	
N	**M**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
100	200	11.448	33.43	8.543	19.097	848	3138
100	400	67.179	118.23	306.863	634.37	35030	70660
300	600	267.643	352.9	107.245	182.76	7505	13478
300	750	1017.37	2210.95	1254.83	2812.6	79855	175564
500	1200	833.533	1506.47	306.508	561.688	24377	32992
500	1300	1879.3	2604.95	1528.88	2716.86	54747	93794

Note that when N or M increases, the number of efficient points and time to generate an efficient point increase significantly. These in turn, increase the time to generate the efficient set. For the networks with (N, M) values of (100, 400), (300, 750) and

(500, 1300), to find all efficient points per instance, we spent approximately 307, 1255 and 1539 seconds on average, respectively. Therefore, we almost triple the total CPU times by enlarging the problem size from (100, 400) to (500, 1300).

Moreover, we also observe a drastic increase both for the CPU times and number of nodes generated when M increases from 200 to 400. Besides the effect of CPU time per efficient point which increases from 0.382 to 1.243, increase on average number of efficient points from 43 to 237 also strengthen the effect and total CPU time ends up with 306.86 seconds on average which is approximately 34 times more than 8.543 seconds. Similarly, increasing the number of arcs by only 200, we increase the average and maximum number of nodes to 34182 and 67522, respectively.

Table 14 and 15 also include the performance statistics of our classical approach. We observe that as N or M increases, the classical approach finds the exact efficient set in much higher solution times.

The significant increases in the average and maximum CPU times can be attributed to the increases in the complexity of the integer models that return a single efficient point and the increase in the number of efficient points.

Note that when N = 100 and M = 200, the average and maximum number of efficient points are 43 and 150 respectively, and time to generate an efficient point is 0.298 seconds on average and maximum 0.49 seconds. 355 average and 503 maximum efficient points are generated (N, M) values of (500, 1300). An efficient point is generated in 5.286 seconds on average and 5.696 seconds at most. Those drastic increases in the number of efficient points and CPU times per efficient point, leads to huge gap between average and maximum CPU times of the networks with (N, M) values of (100, 200) and (500, 1300). All efficient points are generated in 11.448 seconds on average for N = 100, whereas for the network with 500 nodes, average of 1879.3 seconds is spent.

The results of our experiments reveal that the number of arcs does not have significant effect on the performance of classical approach. For example, when N = 100, the average CPU times for M = 200 and 400 are 0.298 and 0.287 seconds, respectively.

Figure 10 compares the CPU times of the branch and bound algorithm and classical approach.

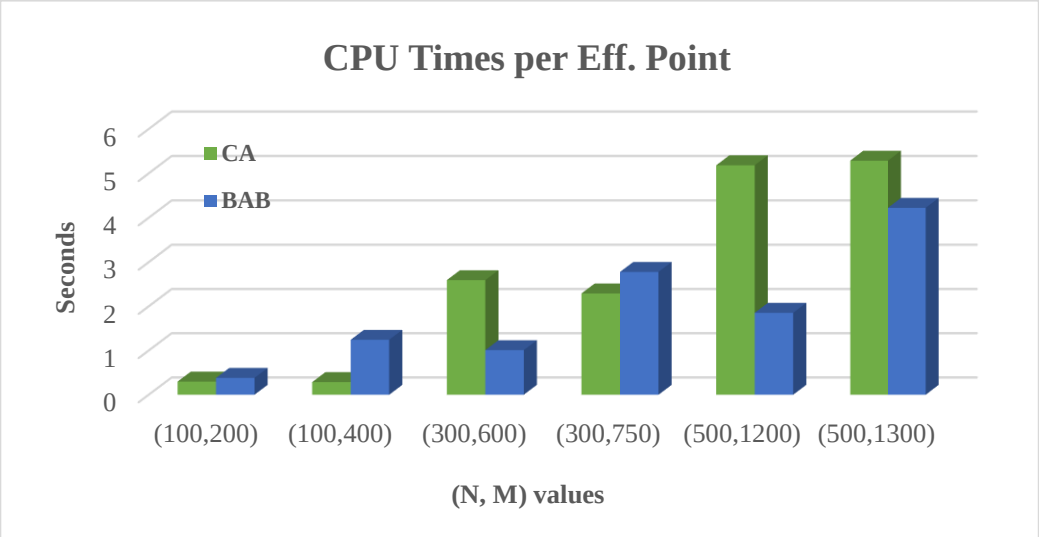


Figure 10: Comparison of CA and BAB Algorithm with respect to CPU Times

Figure 10 has revealed that the branch and bound algorithm outperforms the classical approach (CA) in almost all problem combinations with two exceptions. The first exception is due to the smaller sized problem combination when N = 100 and M = 200. As it seen from Table 14, there is a minor advantage of the classical approach in terms of CPU time per efficient point, which is approximately 0.1 seconds less for the BAB algorithm. The problem size is small so as the number of efficient points is low; therefore, we can see the effect of CPU time spent for the SP rule calculations more clearly on the solution time of an efficient point. The second exception appears when the networks have 400 and 750 arcs. Increasing the gap between number nodes and arcs results in drastic increase in CPU time per efficient point for the BAB algorithm since we perform too many upper bound and lower bound calculations to find an efficient point. For classical approach, on the other hand, we observe almost identical solution times per efficient point when only the number of arcs increases.

The differences between the performances of the BAB algorithm and CA increase as the problem size increases. Note that from the Table 14 when N = 300 and M = 600, the average CPU times per efficient point are 2.593 and 1.01 seconds for classical approach and branch and bound algorithm, respectively. As another notable example, for (N, M) values of (500, 1200), the average CPU times are 5.183 and 1.854 seconds for CA and BAB algorithm, respectively.

We also observe that the BAB algorithm behaves more consistent than classical approach. Note that from the Table 15, for (N, M) values of (500, 1200), the average and maximum CPU times by the BAB algorithm are 306.508 and 561.688 seconds, respectively. For classical approach, we observe the respective average and maximum CPU times of 833.533 and 1506.47 seconds.

CHAPTER 6

CONCLUSIONS

The Chinese Postman Problem (CPP) is a fundamental graph theory problem. It is theoretically important as its solution provides a basis for many Arc Routing Problems (ARPs). Moreover, it has practical appeal as many real world problems can be modeled as CPPs.

The single objective CPP and its variants have been studied extensively for several years. However, to the best of our knowledge, there is no reported work that finds all efficient solutions to the multi-objective CPP.

In this study, we consider a bi-objective CPP. We define a constrained optimization problem and settle its complexity through a reduction to the Generalized Assignment Problem. We use this constrained problem to generate the efficient set with respective two additive objectives, total cost and total distance.

Our approaches to generate the efficient set are the classical approach and the Branch and Bound (BAB) algorithm. We find that our BAB algorithm generates the efficient set for large-sized problem instances with up to 500 nodes and 1300 arcs in less than one hour. The BAB algorithm is found to be superior to the Classical Approach that uses the Mixed Integer Programming (MIP) models.

As a future research, one can propose heuristic procedures that use some truncations of our branch and bound algorithm.

The results of our study can also be extended to the multi-criteria CPP problems with total distance and maximum cost criteria. Considering more than two objectives might be another promising area for future research. Moreover, our results can be extended to other multi-criteria postman problems like Mixed, Rural and Windy Postman Problems.

REFERENCES

Ahr, D., & Reinelt, G. (2002). New heuristics and lower bounds for the Min-Max k-Chinese Postman Problem. In *Algorithms—ESA 2002* (pp. 64-74). Springer Berlin Heidelberg.

Aminu, U. F., & Eglese, R. W. (2006). A constraint programming approach to the Chinese postman problem with time windows. *Computers & Operations Research*, *33*(12), 3423-3431.

Aráoz, J., Fernández, E., & Zoltan, C. (2006). Privatized rural postman problems. *Computers & Operations Research*, *33*(12), 3432-3449.

Aráoz, J., Fernández, E., & Meza, O. (2009). Solving the prize-collecting rural postman problem. *European Journal of Operational Research*, *196*(3), 886-896.

Benavent, E., Campos, V., Corberán, A., & Mota, E. (1992). The capacitated arc routing problem: lower bounds. *Networks*, *22*(7), 669-690.

Brucker, P. (1981). The Chinese postman problem for mixed graphs. In *Graph Theoretic Concepts in Computer Science* (pp. 354-366). Springer Berlin Heidelberg.

Cabral, E. A., Gendreau, M., Ghiani, G., & Laporte, G. (2004). Solving the hierarchical Chinese postman problem as a rural postman problem. *European Journal of Operational Research*, *155*(1), 44-50.

Christofides, N. (1975). *Graph Theory: An Algorithmic Approach*. New York: Academic Press Inc.

Christofides, N., Benavent, E., Campos, V., Corberán, A., & Mota, E. (1984). An optimal method for the mixed postman problem. In *System Modelling and Optimization* (pp. 641-649). Springer Berlin Heidelberg.

Corberán, Á., Plana, I., Rodríguez-Chía, A. M., & Sanchis, J. M. (2013). A branch-and-cut algorithm for the maximum benefit Chinese postman problem. *Mathematical Programming*, *141*(1-2), 21-48.

Corberán, A., Oswald, M., Plana, I., Reinelt, G., & Sanchis, J. M. (2012). New results on the windy postman problem. *Mathematical Programming*, *132*(1-2), 309-332.

Corberán, A., & Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, *56*(1), 50-69.

Degenhardt, J. (2004). An ant-algorithm for the balanced k-chinese postmen problem. In *Operations Research Conference*.

Dror, M., & Haouari, M. (2000). Generalized Steiner problems and other variants. *Journal of Combinatorial Optimization*, *4*(4), 415-436.

Dror, M., Stern, H., & Trudeau, P. (1987). Postman tour on a graph with precedence relation on arcs. *Networks*, *17*(3), 283-294.

Edmonds, J. (1965, January). Chinese Postmen Problem. In *Operations Research* (p. B73). 901 Elkridge Landing Rd, Ste 400, Linthicum Hts, md 21090-2909: Inst Operations Research Management Sciences.

Edmond, J., & Johnson, E. L. (1973). Matching, Euler Tour and The Chinese Postman Problem. *Mathematical Programming*, *5*, 88-124.

Eiselt, H. A., Gendreau, M., & Laporte, G. (1995). Arc routing problems, part I: The Chinese postman problem. *Operations Research*, *43*(2), 231-242.

Euchi, J., Chabchoub, H., & Yassine, A. (2010, May). Metaheuristics optimization via memory to solve the profitable arc tour problem. In *8th International Conference of Modeling and Simulation*.

Evans, J. (1992). Optimization algorithms for networks and graphs. CRC Press.

Feillet, D., Dejax, P., & Gendreau, M. (2005). The profitable arc tour problem: solution with a branch-and-price algorithm. *Transportation Science*, *39*(4), 539-552.

Ford, L. R., & Fulkerson, D. R. (1962). *Flows in networks* (Vol. 1962). Princeton University Press: Princeton.

Frederickson, G. N., Hecht, M. S., & Kim, C. E. (1976, October). Approximation algorithms for some routing problems. In *Foundations of Computer Science, 1976., 17th Annual Symposium on* (pp. 216-227). IEEE.

Ghiani, G., & Improta, G. (2000). An algorithm for the hierarchical Chinese postman problem. *Operations Research Letters*, *26*(1), 27-32.

Grötschel, M., & Win, Z. (1992). A cutting plane algorithm for the windy postman problem. *Mathematical Programming*, *55*(1-3), 339-358.

Guan, M. (1962). Graphic programming using odd or even points. *Chinese Math*, 1(273-277), 110.

Haimes, Y. Y., Lasdon, L. S., & Wismer, D. A. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems Man and Cybernetics*, (1), 296-297.

Kaufmann, A. (1967). Graphs, dynamic programming, and finite games (Vol. 36). New York: Academic Press.

Korteweg, P., & Volgenant, T. (2006). On the Hierarchical Chinese Postman Problem with linear ordered classes. *European Journal of Operational Research,169*(1), 41-52.

Malandraki, C., & Daskin, M. S. (1993). The maximum benefit Chinese postman problem and the maximum benefit traveling salesman problem. *European Journal of Operational Research*, *65*(2), 218-234.

Martello, S., & Toth, P. (1990). An exact algorithm for large unbounded knapsack problems. *Operations Research Letters*, 9(1), 15-20.

Minieka, E. (1979). The Chinese postman problem for mixed networks. *Management Science*, *25*(7), 643-648.

Nobert, Y. (1991). An optimal algorithm for the mixed Chinese postman problem. *Centre de Recherche Sur Les Transports Publication*, (799).

Papadimitriou, C. H. (1976). On the complexity of edge traversing. *Journal of the ACM (JACM)*, *23*(3), 544-554.

Pearn, W. L. (1994). Solvable cases of the k-person Chinese postman problem. *Operations Research Letters*, *16*(4), 241-244.

Pearn, W. L. (1991). On the k-person Chinese postman problem. *Technical Report, National Chiao Tung University, Hsinchu, Taiwan, ROC*.

Pearn, W. L., & Chiu, W. C. (2005). Approximate solutions for the maximum benefit Chinese postman problem. *International Journal of Systems Science,36*(13), 815-822.

Pearn, W. L., & Wang, K. H. (2003). On the maximum benefit Chinese postman problem. *Omega*, *31*(4), 269-273.

Prakash, S., Sharma, M. K., & Singh, A. (2009, July). A heuristic for multi-objective Chinese postman problem. In *Computers & Industrial Engineering, 2009. CIE 2009. International Conference on* (pp. 596-599). IEEE.

Thimbleby, H. (2003). The directed Chinese postman problem. *Software: Practice and Experience*, *33*(11), 1081-1096.

Thomassen, C. (1997). On the complexity of finding a minimum cycle cover of a graph. *SIAM Journal on Computing*, *26*(3), 675-677.

Win, Z. (1989). On the windy postman problem on Eulerian graphs. *Mathematical Programming*, *44*(1-3), 97-112.

APPENDIX A

ALL SHORTEST PATHS ACCORDING TO EACH ARC WEIGHT FOR THE INSTANCE II

Table 16**:** All Shortest Paths According to Each Arc Weight for the Instance II

[i, j]	**Shortest Paths wrt c_{ij}**	**Shortest Paths wrt d_{ij}**
[6 - 1]	6-29-36-27-20-19-1	6-29-36-27-20-19-1
[6 - 4]	6-29-36-27-20-19-21-22-26-4	6-29-36-27-20-19-21-22-26-4
[6 - 5]	6-29-36-17-15-16-7-5	6-29-36-27-20-19-1-7-5
[6 - 14]	6-29-36-27-20-19-14	6-29-36-27-20-19-14
[6 - 15]	6-29-36-17-15	6-29-36-17-15
[6 - 17]	6-29-36-17	6-29-36-17
[6 - 19]	6-29-36-27-20-19	6-29-36-27-20-19
[6 - 21]	6-29-36-27-20-19-21	6-29-36-27-20-19-21
[6 - 22]	6-29-36-27-20-19-21-22	6-29-36-27-20-19-21-22
[6 - 23]	6-29-36-27-20-19-21-22-23	6-29-36-27-20-19-21-22-23
[6 - 26]	6-29-36-27-20-19-21-22-26	6-29-36-27-20-19-21-22-26
[6 - 27]	6-29-36-27	6-29-36-27
[6 - 28]	6-29-28	6-29-28
[6 - 29]	6-29	6-29
[6 - 33]	6-29-10-31-33	6-29-10-31-33
[8 - 1]	8-6-29-36-27-20-19-1	8-6-29-36-27-20-19-1
[8 - 4]	8-6-29-36-27-20-19-21-22-26-4	8-6-29-36-27-20-19-21-22-26-4
[8 - 5]	8-6-29-36-17-15-16-7-5	8-6-29-36-27-20-19-1-7-5
[8 - 14]	8-6-29-36-27-20-19-14	8-6-29-36-27-20-19-14
[8 - 15]	8-6-29-36-17-15	8-6-29-36-17-15
[8 - 17]	8-6-29-36-17	8-6-29-36-17
[8 - 19]	8-6-29-36-27-20-19	8-6-29-36-27-20-19
[8 - 21]	8-6-29-36-27-20-19-21	8-6-29-36-27-20-19-21
[8 - 22]	8-6-29-36-27-20-19-21-22	8-6-29-36-27-20-19-21-22
[8 - 23]	8-6-29-36-27-20-19-21-22-23	8-6-29-36-27-20-19-21-22-23
[8 - 26]	8-6-29-36-27-20-19-21-22-26	8-6-29-36-27-20-19-21-22-26

Table 16 (cont'd): All Shortest Paths According to Each Arc Weight for the Instance II

[i, j]	Shortest Paths wrt c_{ij}	Shortest Paths wrt d_{ij}
[8 - 27]	8-6-29-36-27	8-6-29-36-27
[8 - 28]	8-6-29-28	8-6-29-28
[8 - 29]	8-6-29	8-6-29
[8 - 33]	8-6-29-10-31-33	8-6-29-10-31-33
[9 - 1]	9-29-36-27-20-19-1	9-29-36-27-20-19-1
[9 - 4]	9-29-36-27-20-19-21-22-26-4	9-29-36-27-20-19-21-22-26-4
[9 - 5]	9-29-36-17-15-16-7-5	9-29-36-27-20-19-1-7-5
[9 - 14]	9-29-36-27-20-19-14	9-29-36-27-20-19-14
[9 - 15]	9-29-36-17-15	9-29-36-17-15
[9 - 17]	9-29-36-17	9-29-36-17
[9 - 19]	9-29-36-27-20-19	9-29-36-27-20-19
[9 - 21]	9-29-36-27-20-19-21	9-29-36-27-20-19-21
[9 - 22]	9-29-36-27-20-19-21-22	9-29-36-27-20-19-21-22
[9 - 23]	9-29-36-27-20-19-21-22-23	9-29-36-27-20-19-21-22-23
[9 - 26]	9-29-36-27-20-19-21-22-26	9-29-36-27-20-19-21-22-26
[9 - 27]	9-29-36-27	9-29-36-27
[9 - 28]	9-29-28	9-29-28
[9 - 29]	9-29	9-29
[9 - 33]	9-29-10-31-33	9-29-10-31-33
[10 - 1]	10-31-33-36-27-20-19-1	10-31-33-36-27-20-19-1
[10 - 4]	10-31-33-36-27-20-19-21-22-26-4	10-31-33-36-27-20-19-21-22-26-4
[10 - 5]	10-31-33-36-17-15-16-7-5	10-31-33-36-27-20-19-1-7-5
[10 - 14]	10-31-33-36-27-20-19-14	10-31-33-36-27-20-19-14
[10 - 15]	10-31-33-36-17-15	10-31-33-36-17-15
[10 - 17]	10-31-33-36-17	10-31-33-36-17
[10 - 19]	10-31-33-36-27-20-19	10-31-33-36-27-20-19
[10 - 21]	10-31-33-36-27-20-19-21	10-31-33-36-27-20-19-21
[10 - 22]	10-31-33-36-27-20-19-21-22	10-31-33-36-27-20-19-21-22
[10 - 23]	10-31-33-36-27-20-19-21-22-23	10-31-33-36-27-20-19-21-22-23
[10 - 26]	10-31-33-36-27-20-19-21-22-26	10-31-33-36-27-20-19-21-22-26
[10 - 27]	10-31-33-36-27	10-31-33-36-27
[10 - 28]	10-31-33-36-27-9-29-28	10-31-33-36-17-11-29-28
[10 - 29]	10-31-33-36-27-9-29	10-31-33-36-17-11-29
[10 - 33]	10-31-33	10-31-33
[11 - 1]	11-29-36-27-20-19-1	11-29-36-27-20-19-1
[11 - 4]	11-29-36-27-20-19-21-22-26-4	11-29-36-27-20-19-21-22-26-4
[11 - 5]	11-29-36-17-15-16-7-5	11-29-36-27-20-19-1-7-5

Table 16 (cont'd): All Shortest Paths According to Each Arc Weight for the Instance II

[i, j]	Shortest Paths wrt c_{ij}	Shortest Paths wrt d_{ij}
[11 - 14]	11-29-36-27-20-19-14	11-29-36-27-20-19-14
[11 - 15]	11-29-36-17-15	11-29-36-17-15
[11 - 17]	11-29-36-17	11-29-36-17
[11 - 19]	11-29-36-27-20-19	11-29-36-27-20-19
[11 - 21]	11-29-36-27-20-19-21	11-29-36-27-20-19-21
[11 - 22]	11-29-36-27-20-19-21-22	11-29-36-27-20-19-21-22
[11 - 23]	11-29-36-27-20-19-21-22-23	11-29-36-27-20-19-21-22-23
[11 - 26]	11-29-36-27-20-19-21-22-26	11-29-36-27-20-19-21-22-26
[11 - 27]	11-29-36-27	11-29-36-27
[11 - 28]	11-29-28	11-29-28
[11 - 29]	11-29	11-29
[11 - 33]	11-29-10-31-33	11-29-10-31-33
[24 - 1]	24-9-29-36-27-20-19-1	24-9-29-36-27-20-19-1
[24 - 4]	24-9-29-36-27-20-19-21-22-26-4	24-9-29-36-27-20-19-21-22-26-4
[24 - 5]	24-9-29-36-17-15-16-7-5	24-9-29-36-27-20-19-1-7-5
[24 - 14]	24-9-29-36-27-20-19-14	24-9-29-36-27-20-19-14
[24 - 15]	24-9-29-36-17-15	24-9-29-36-17-15
[24 - 17]	24-9-29-36-17	24-9-29-36-17
[24 - 19]	24-9-29-36-27-20-19	24-9-29-36-27-20-19
[24 - 21]	24-9-29-36-27-20-19-21	24-9-29-36-27-20-19-21
[24 - 22]	24-9-29-36-27-20-19-21-22	24-9-29-36-27-20-19-21-22
[24 - 23]	24-9-29-36-27-20-19-21-22-23	24-9-29-36-27-20-19-21-22-23
[24 - 26]	24-9-29-36-27-20-19-21-22-26	24-9-29-36-27-20-19-21-22-26
[24 - 27]	24-9-29-36-27	24-9-29-36-27
[24 - 28]	24-9-29-28	24-9-29-28
[24 - 29]	24-9-29	24-9-29
[24 - 33]	24-9-29-10-31-33	24-9-29-10-31-33
[25 - 1]	25-33-36-27-20-19-1	25-33-36-27-20-19-1
[25 - 4]	25-33-36-27-20-19-21-22-26-4	25-33-36-27-20-19-21-22-26-4
[25 - 5]	25-33-36-17-15-16-7-5	25-33-36-27-20-19-1-7-5
[25 - 14]	25-33-36-27-20-19-14	25-33-36-27-20-19-14
[25 - 15]	25-33-36-17-15	25-33-36-17-15
[25 - 17]	25-33-36-17	25-33-36-17
[25 - 19]	25-33-36-27-20-19	25-33-36-27-20-19
[25 - 21]	25-33-36-27-20-19-21	25-33-36-27-20-19-21
[25 - 22]	25-33-36-27-20-19-21-22	25-33-36-27-20-19-21-22
[25 - 23]	25-33-36-27-20-19-21-22-23	25-33-36-27-20-19-21-22-23
[25 - 26]	25-33-36-27-20-19-21-22-26	25-33-36-27-20-19-21-22-26

Table 16 (cont'd): All Shortest Paths According to Each Arc Weight for the Instance II

[i, j]	Shortest Paths wrt c_{ij}	Shortest Paths wrt d_{ij}
[25 - 27]	25-33-36-27	25-33-36-27
[25 - 28]	25-33-36-27-9-29-28	25-33-36-17-11-29-28
[25 - 29]	25-33-36-27-9-29	25-33-36-17-11-29
[25 - 33]	25-33	25-33
[31 - 1]	31-33-36-27-20-19-1	31-33-36-27-20-19-1
[31 - 4]	31-33-36-27-20-19-21-22-26-4	31-33-36-27-20-19-21-22-26-4
[31 - 5]	31-33-36-17-15-16-7-5	31-33-36-27-20-19-1-7-5
[31 - 14]	31-33-36-27-20-19-14	31-33-36-27-20-19-14
[31 - 15]	31-33-36-17-15	31-33-36-17-15
[31 - 17]	31-33-36-17	31-33-36-17
[31 - 19]	31-33-36-27-20-19	31-33-36-27-20-19
[31 - 21]	31-33-36-27-20-19-21	31-33-36-27-20-19-21
[31 - 22]	31-33-36-27-20-19-21-22	31-33-36-27-20-19-21-22
[31 - 23]	31-33-36-27-20-19-21-22-23	31-33-36-27-20-19-21-22-23
[31 - 26]	31-33-36-27-20-19-21-22-26	31-33-36-27-20-19-21-22-26
[31 - 27]	31-33-36-27	31-33-36-27
[31 - 28]	31-33-36-27-9-29-28	31-33-36-17-11-29-28
[31 - 29]	31-33-36-27-9-29	31-33-36-17-11-29
[31 - 33]	31-33	31-33
[32 - 1]	32-31-33-36-27-20-19-1	32-31-33-36-27-20-19-1
[32 - 4]	32-31-33-36-27-20-19-21-22-26-4	32-31-33-36-27-20-19-21-22-26-4
[32 - 5]	32-31-33-36-17-15-16-7-5	32-31-33-36-27-20-19-1-7-5
[32 - 14]	32-31-33-36-27-20-19-14	32-31-33-36-27-20-19-14
[32 - 15]	32-31-33-36-17-15	32-31-33-36-17-15
[32 - 17]	32-31-33-36-17	32-31-33-36-17
[32 - 19]	32-31-33-36-27-20-19	32-31-33-36-27-20-19
[32 - 21]	32-31-33-36-27-20-19-21	32-31-33-36-27-20-19-21
[32 - 22]	32-31-33-36-27-20-19-21-22	32-31-33-36-27-20-19-21-22
[32 - 23]	32-31-33-36-27-20-19-21-22-23	32-31-33-36-27-20-19-21-22-23
[32 - 26]	32-31-33-36-27-20-19-21-22-26	32-31-33-36-27-20-19-21-22-26
[32 - 27]	32-31-33-36-27	32-31-33-36-27
[32 - 28]	32-31-33-36-27-9-29-28	32-31-33-36-17-11-29-28
[32 - 29]	32-31-33-36-27-9-29	32-31-33-36-17-11-29
[32 - 33]	32-31-33	32-31-33
[35 - 1]	35-36-27-20-19-1	35-36-27-20-19-1
[35 - 4]	35-36-27-20-19-21-22-26-4	35-36-27-20-19-21-22-26-4

Table 16 (cont'd): All Shortest Paths According to Each Arc Weight for the Instance II

[i, j]	Shortest Paths wrt c_{ij}	Shortest Paths wrt d_{ij}
[35 - 5]	35-36-17-15-16-7-5	35-36-27-20-19-1-7-5
[35 - 14]	35-36-27-20-19-14	35-36-27-20-19-14
[35 - 15]	35-36-17-15	35-36-17-15
[35 - 17]	35-36-17	35-36-17
[35 - 19]	35-36-27-20-19	35-36-27-20-19
[35 - 21]	35-36-27-20-19-21	35-36-27-20-19-21
[35 - 22]	35-36-27-20-19-21-22	35-36-27-20-19-21-22
[35 - 23]	35-36-27-20-19-21-22-23	35-36-27-20-19-21-22-23
[35 - 26]	35-36-27-20-19-21-22-26	35-36-27-20-19-21-22-26
[35 - 27]	35-36-27	35-36-27
[35 - 28]	35-36-27-9-29-28	35-36-17-11-29-28
[35 - 29]	35-36-27-9-29	35-36-17-11-29
[35 - 33]	35-36-27-9-29-10-31-33	35-36-27-20-19-1-25-33
[36 - 1]	36-27-20-19-1	36-27-20-19-1
[36 - 4]	36-27-20-19-21-22-26-4	36-27-20-19-21-22-26-4
[36 - 5]	36-17-15-16-7-5	36-27-20-19-1-7-5
[36 - 14]	36-27-20-19-14	36-27-20-19-14
[36 - 15]	36-17-15	36-17-15
[36 - 17]	36-17	36-17
[36 - 19]	36-27-20-19	36-27-20-19
[36 - 21]	36-27-20-19-21	36-27-20-19-21
[36 - 22]	36-27-20-19-21-22	36-27-20-19-21-22
[36 - 23]	36-27-20-19-21-22-23	36-27-20-19-21-22-23
[36 - 26]	36-27-20-19-21-22-26	36-27-20-19-21-22-26
[36 - 27]	36-27	36-27
[36 - 28]	36-27-9-29-28	36-17-11-29-28
[36 - 29]	36-27-9-29	36-17-11-29
[36 - 33]	36-27-9-29-10-31-33	36-27-20-19-1-25-33